The Most Comprehensive Reference Book on the Apple II Computer Series Including IIc

APPLE THESAURUS

by Aaron Filler

APPLE THESAURUS

by
Aaron Filler

with
Nick Anis
and
Terry Deacon

DATAMOST™
20660 Nordhoff Street, Chatsworth, CA 91311-6152
(818) 709-1202

ISBN 0-88190-346-9

Printed in U.S.A.

Table of Contents

Illustrations

The following is a list of manufacturers and individuals who provided photographs, diagrams and other information contained within this book.

Abacus
25.4, 27.2

ABT
8.7, 8.10, 8.11

Abacus
25.4, 27.2

ABT
8.7, 8.10, 8.11

Amdek
7.2a, 7.5, 24.7

Analytical Engines
30.3

Apple Computer
2.1, 5.4a, 5.4b, 6.1, 8.1a, 8.1b, 9.0, 9.7, 12.2a, 17.9, 19.8, 23.5, 24.1, 24.4a, 24.11, 32.0, 42.1, 43.1

Appendices
A, B, C, and H

Call A.P.P.L.E. (Charles Kluepfel)
Listing 40.3

CCS
14.2

Nancy Cleveland
1.0, 1.1a, 1.1b, 1.2, 1.3, 1.4a, 1.5, 1.6a, 2.0, 2.4a, 2.4b, 2.5, 3.0, 4.0, 4.1, 4.2, 5.0, 5.2, 5.5, 5.6, 5.7a, 5.7b, 5.8a, 5.8b, 6.0, 6.2a, 6.3 6.5a, 6.5b, 7.0, 7.1a, 7.4, 8.3a, 8.4, 8.5, 8.6, 8.9, 9.9, 11.0, 12.0, 12.1, 12.3, 12.4, 13.0, 13.1, 13.4, 14.0, 14.1c, 14.3, 14.5, 15.0, 15.1a, 16.0, 16.3a, 17.0, 17.6, 17.8, 18.0, 18.3, 18.6, 18.8, 19.0, 19.2, 19.5, 20.0, 21.0, 21.8, 21.9,21.11, 21.12, 21.14a, 22.0, 22.2, 22.3, 22.4, 22.6, 22.7, 22.8, 23.0, 23.1a, 23.4b, 23.7, 23.8, 23.9, 24.0, 24.8, 24.10, 24.14, 24.17, 25.3, 26.0, 26.2a, 26.2b, 26.8, 28.0, 29.0, 29.2, 29.4, 29.6, 29.7, 30.0, 31.0, 31.1a, 31.1b, 33.0, 34.0, 35.0, 36.0, 37.0, 37.1, 38.0, 39.0, 40.0, 41.0

Cyborg
15.3, 15.5

Data Translation
15.2

Davong
24.12, 24.13

Enter
19.7

Epson
19.1, 19.3

Hayes
17.3, 17.10

HP
19.6, 19.9

IDT
24.14

Interactive Structures
15.1b, 15.6, 18.2

Keithley-DAS
15.4

Keytronic
8.8

Koala
9.2

Kraft
9.4a, 9.4b

Legend
35.0

Magellan
9.1

M&R
12.5, 12.8

Microsci
24.5

David H. Miller
43.0

MOS Technology
27.1

MPC
18.12b, 24.16

Novation
17.4, 17.11, 17.12

Orange
18.7, 18.10, 18.11

Practical Peripherals
18.9, 18.12a

Qume
2.3, 18.1, 19.4, 24.3b

Rana
23.3, 24.2, 24.6, 24.9, 30.2

RH Electronics
12.6, 12.7a, 12.7b, 12.9

Softalk
Listing 7.2

SSM
18.5

Synertek
16.1, 27.0, 27.1, 28.1, 28.2

Synetix
7.3

Syntauri
10.5

Taxan
7.6

TG
9.4c, 9.4d, 9.5, 9.6

Titan
25.5

Versa
9.8

Videx
6.4

Voice Machine Communications
11.2

Votrax Division of Federal Screw Works
11.1a, 11.1b

Western Design Center
Table 29.1a, Table 29.1b

Figure 42.0 provided by Steven Wozniak.
All photographs, drawings, and tables not listed above are by A. Filler.

Preface

If you're in the mood for launching into a full scale, eight hundred page reading binge, then you will benefit from the chosen sequence of information in the book. If you're only up for one or two hundred pages on your chosen subject, then you might do well to read an entire part in order. However, each chapter stands by itself, and within each chapter there are enough summaries and cross references that you should be able to plunge in at will just about anywhere.

A lot of effort went into creating a very detailed index which should help you get quick information on specific kinds of products, words you're not familiar with, and particular aspects of the Apple's hardware and system software. The bulk of information in the book pertains to the II+, //e and //c computers, but the old II also gets considerable coverage.

A key point for many readers to know in advance is that the technical level of the text varies. The usual pattern is to begin with a full fledged technical explanation, then to follow with a more general summary, and then to slip on up to the most general level when actually comparing products. So if you don't understand things where you're reading, try another paragraph.

The book doesn't offer sufficient technical detail for a determined assembly language programmer in training or erstwhile hardware designer. The objective for such folks is only to explain what is going on and to point you in the direction of the appropriate technical source.

The intended audience is the Apple user who just finds all this stuff interesting and wants to get a handle on how things work.

None of the authors are electrical engineers or professional programmers.

Aaron Filler, who organized and wrote this book, is a M.D./Ph.D. student who has spent the past five years doing research and teaching in biology and human evolution at Harvard University. He is currently completing his medical training at the University of Chicago. He has experience in assembling and programming Apple-based laboratory systems.

Nick Anis, who arranged all of the contacts with information sources and collected most of the research information, is a consultant in Fullerton, California who has worked as a technical support director for a large manufacturer of Apple peripherals.

Terry Deacon, who carried out or supervised most of the hardware evaluations, is an instructor at Harvard University who uses microcomputers in his neurobiology research.

In the course of preparing the book we received valuable information and assistance from many people at Apple Computer: Erika Vogler provided photographs and arranged contacts, Peter Baum and Steve Wozniak discussed hardware, and Ernie Beernink, Dick Houston, and Bill Schjelderup talked to us about operating system software. Special thanks go to John Sculley and Steve Wozniak of Apple, for participating in the comments section.

We also thank Pat Caffrey for explaining the Apple interrupt bug, William Mensch and David Eyes for discussing the 65816 and 65802, Jamie Taffe of Pion for talking about RAM, Wink Saville and Bill Graves for information on a few software mysteries, Matt Filler who made available his tables of printer escape codes, and Alan Roskiewicz of A+ magazine with whom we pooled our evaluation models of speech synthesis and recognition products.

Ron Davis and Joel Mode of the The Computer Store in Cambridge, Massachusetts, Tom Freeman of ACP in Santa Ana, California, and Jan McGowan of MicroPro provided access to technical literature and hardware at crucial moments. Eli Heffron's in Cambridge, Massachusetts permitted us to photograph electronic components and computer systems in their store.

More than two hundred companies sent us detailed technical literature and photographs which provided the basis for our discussions of nearly 800 peripheral products. Most of the companies readily put engineers on the phone to explain their products to us and a great many provided products for testing and evaluation.

We gratefully thank the hundreds of sales and technical people we dealt with at the various companies and we also thank Ron Beekley who did the dBase II programming to help us keep track of all this product information, Patty Anis who entered much of the data, and the numerous computer users among the students and faculty at Harvard as well as in Fullerton who helped us in product evaluations.

We thank David H. Miller for taking the time to do a photo session with John Sculley which resulted in the excellent shot in Chapter 43. Among the chapter front pieces and product shots are over a hundred photographs by Nancy Cleveland, selected from the more than 500 photos she took for this project.

In the course of preparing the manuscript, we had the benefit of thoughtful comments and suggestions from Eugene Altmann, Dr. Richard Smith, and Pete McDonald all of whom read drafts of the book, as well as DATAMOST editors Scott Wilson, Marcia Carrozzo, and Lorraine Coffey who saw the book through to completion.

Finally, we're counting on the readers of this first edition to tell us about aspects of the Apple you'd like to see explained in more detail or about which you have special knowledge. We apologize in advance for any errors, omissions, or murky explanations that may have crept in. Comments, suggestions, or information on new products should be sent to: Apple Thesaurus, 524 W. Commonwealth Ave., Suite M, Fullerton, CA. 92632.

I

INTRODUCTORY APPLE

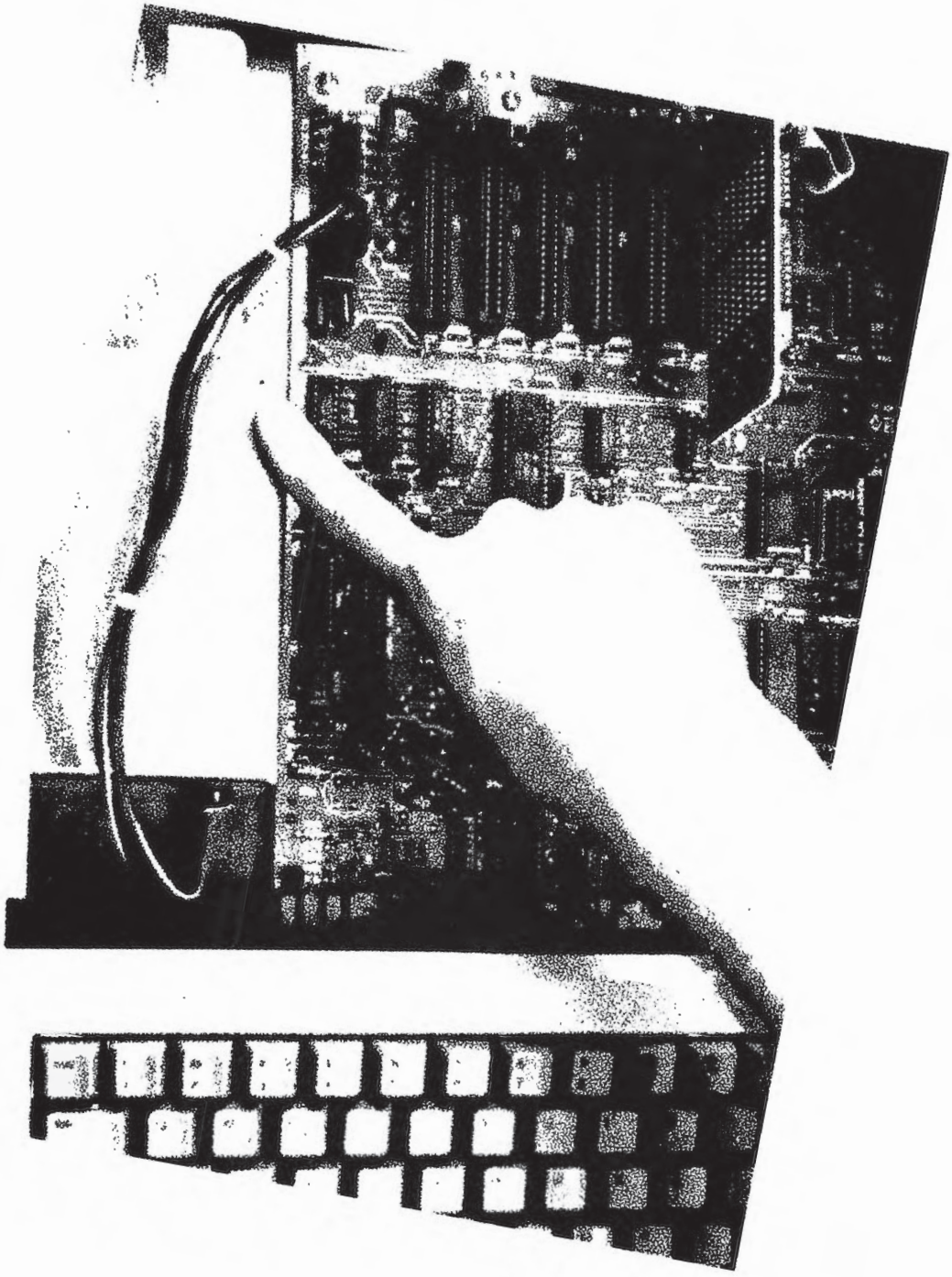
PART 1 ..... The Beginning Apple

PART 2 ..... The Advanced Apple

PART 1

The Beginning Apple

- **CHAPTER 1 Processing and Memory**
- **CHAPTER 2 Output, Input, and Expansion**



Chapter 1

Processing and Memory

Inside The Apple

Turn off the Apple's power and rip the top off.

Your dealer will let you do this in the showroom, and you certainly can do it at home where no one's watching (except if you have a //c, of course, in which case DO NOT rip the top off, but do look at Figure 1.1 where we rip the top off for you). You're now looking down onto the stuff of tomorrow's dreams—epoxy, silicon, copper and plain old plastic. With power off but cord still plugged in, look into the Apple and with one finger reach in and touch the large golden metal box on the left. Pause, then remove your hand from the Apple. A bizarre new cult ritual? Perhaps "yes" in some seldom traveled pockets of Southern California, but for the most part not really. You have just discharged your static electricity.

Static is one of the four Apple frying entities you'll be learning to watch out for. It can cause strange behavior in an otherwise well-behaved Apple and damage new computer parts as well as cause embarrassment during handshakes.

Fortunately, static electric charges seek out the path of least resistance to the earth. Your favorite path will be from your hand to the metal box to the power cord, down to your basement, and on to any water pipe to ground (meaning exactly "the earth itself"). This is the only point at which plumbing is directly relevant to your Apple.

The large green card that fills most of the Apple is called the motherboard. The first thing that may strike you as unusual about such a complicated electronic device is that there aren't any wires (except for that one lonely set that connects to the box on the left). This is OK, as modern computers don't use wires. It's all done with cameras. To wit, photolithography in sheets of copper on panels of epoxy has completely replaced the familiar wires of TVs, radios and stereos of the recent past.

Scattered about the motherboard are small black plastic rectangles on metal legs. These are Dual In-line Packages (DIPs—pronounced either like "zip" or like V.I.P., depending on your mood and which sounds best in the particular phrase at hand), the electronic "hardware" of which your computer is made. Some of these have eight legs, some have 14, but the grand master chips of this computer generation are the 40 pin DIPs. In the //e and //c there are four of these, in the II and II+ there is only one. In either case, in every Apple there is at least one 40 pin DIP with the numbers 6502 printed somewhere on its back (see Figure 1.1). Inside this one single package is the Apple incarnate.

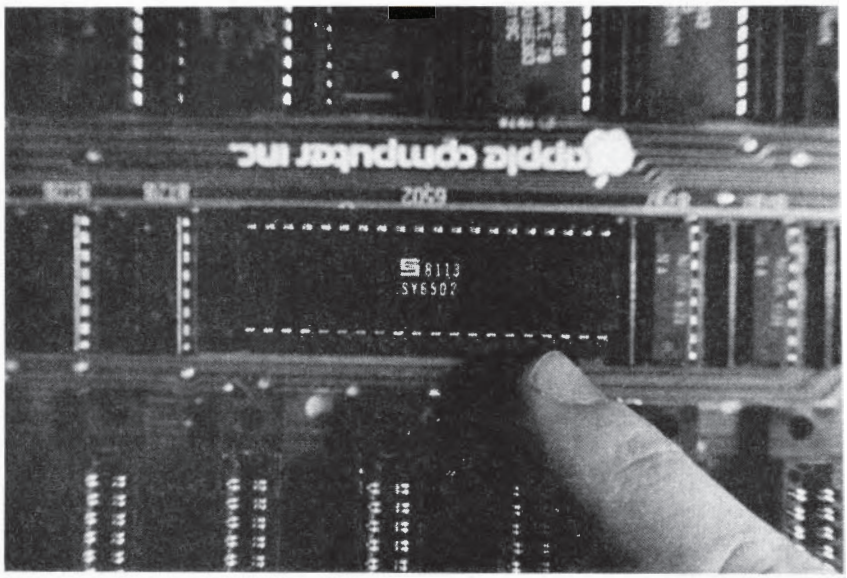


Fig. 1.1a The 6502 on the II/II+ motherboard. It is the largest DIP on the Apple II/II+. The numbers 6502 are printed on the top.

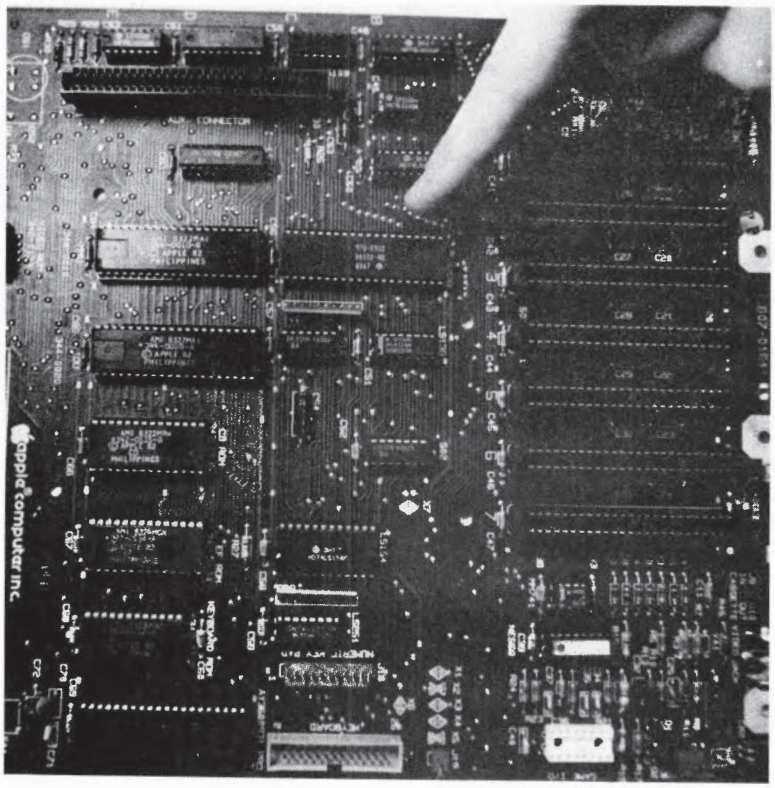


Fig. 1.1b The 6502 on the //e motherboard. There are four 40-pin DIPs on the //e motherboard.

Fig. 1.1c (right) Under the cover of the //c: keyboard, disk drive, and internal power converter.

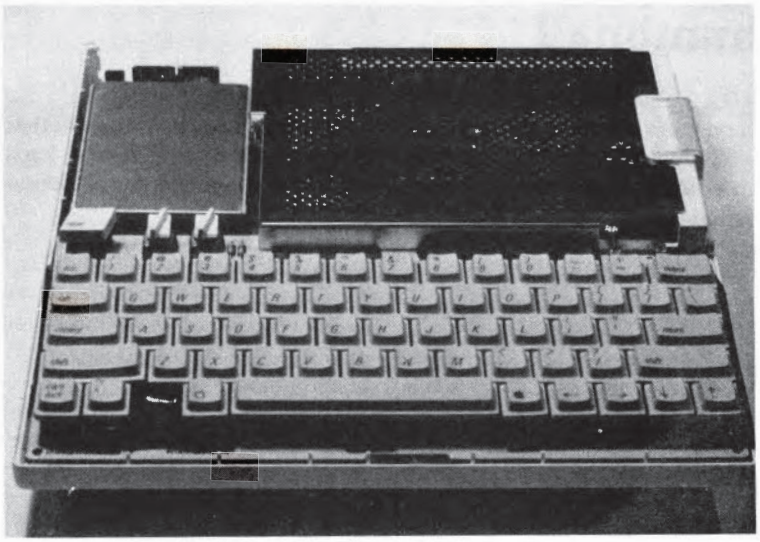
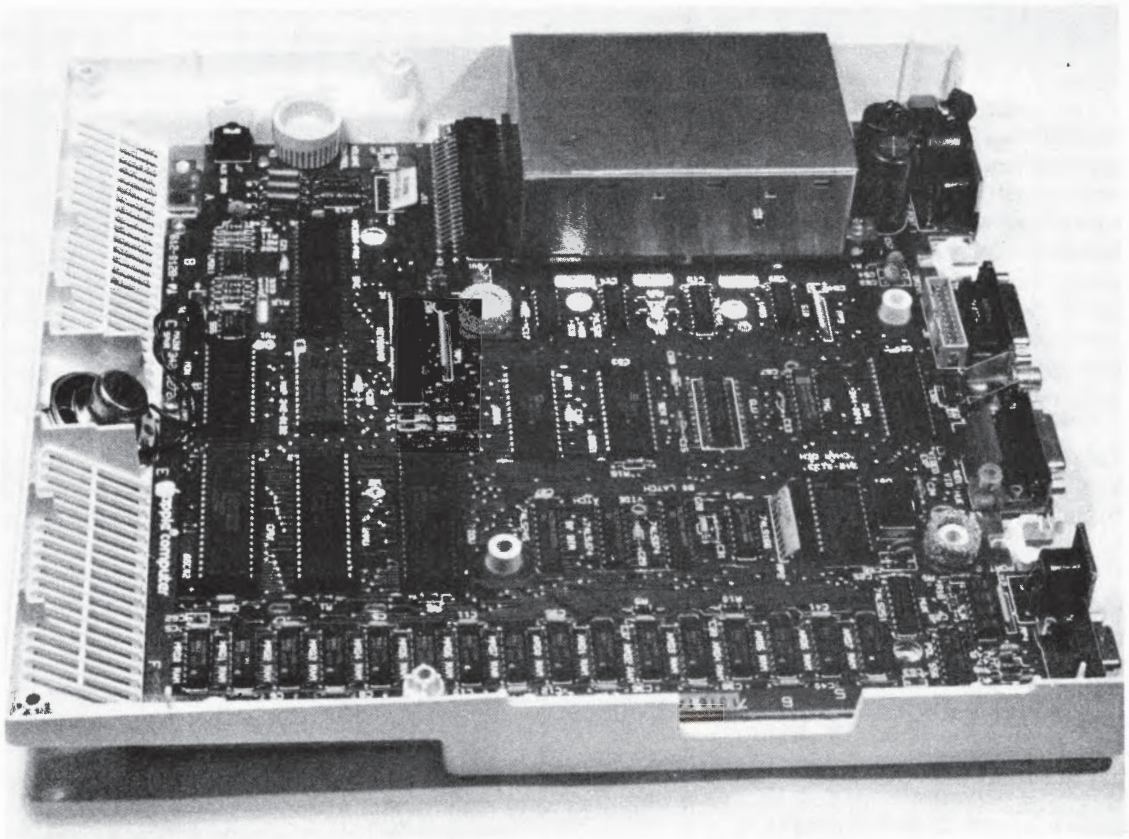


Fig. 1.1d (below) The //c motherboard. Like the //e, the //c has four 40-pin DIPs.



The Apple's Microprocessor: Apple on a Thumbnail

The 6502 plastic package conceals and contains one tiny flake of silicon about a quarter of an inch on each side (see Figure 1.2). The package and the chip together cost all of about $3.75. The $2000 worth of odds and ends that make up the rest of your Apple exist only to serve the momentary whims and wants of the tiny 6502 silicon microprocessor chip here ensconced unceremoniously in dull black plastic.

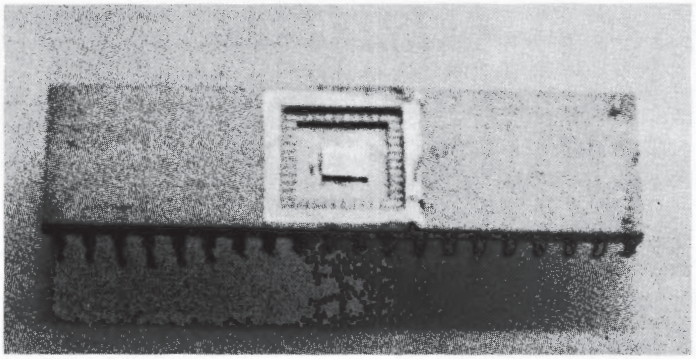


Fig. 1.2 The silicon chip is the smallest square in the center of the 40 pin DIP.

The mind of an Apple, embodied in a flesh of silicon called the 6502, has two key characteristics: stupefying simplicity and staggering speed. The most elaborate thought that a 6502 is capable of runs something like this: "Get the number at address 31,325; add one to the number; put the number back at address 31,325; get the number at address 31,326, etc.," ad infinitum. The trick is that the 6502 can step through its simplest thoughts in about one millionth of a second. Really complex thoughts may take 10 millionths of a second. When 10 seconds have gone by, your Apple may have done two or three million separate little things. Now each of these little things taken alone are not very impressive; but let's face it, two or three million of just about anything is sufficiently into the gobs and gobs range that it just can't be ignored or laughed at without a second glance.

Later on we'll take a look at the world as seen through the eyes of the 6502, but for now, just take a moment to contemplate in the abstract what a ridiculous yet exquisite wonder that man has wrought in this odd new machine.

Apple Memory: What it is and What is in it

The Nature of Computer Memory: How and What to Remember

What about all the other DIP chips out there? A lot of that stuff has to do with memory. The most critical memory in the Apple resides in a few very small areas within the 6502 itself. In the best of all possible worlds, all of the Apple's memory would reside inside the 6502, but in the 1980s this is not yet physically possible. The average Apple has over 500,000 little tiny memory units called "dynamic storage cells." Little though they may be, you'd have to put

each one into a space one half millionth of an inch across if you wanted to put them all in the 6502—this just can't be done. Therefore they end up scattered around the motherboard in various assorted chips, and a substantial array of electrical connections and switching mechanisms are required to fool the 6502 into believing that they're right inside the DIP along with it.

Memory is used for two things in a computer. One is to hold computer programs (the verbs and articles of computer function), and the other is to hold data (the nouns).

A computer program is a series of instructions which tell the 6502 what to do next (i.e., add two numbers, write some number to some address in memory, or read the next program instruction). To operate the 6502, a computer programmer puts a list of such instructions into the Apple's memory somewhere, tells the 6502 the "address" of the location in memory of the first instruction on the list, and then tells the 6502 to start taking its instructions, in order, from that list. And we do mean a *programmer.* You will never have to think about your Apple this way unless you want to.

Most Apple users buy complete program packages such as word processing packages or accounting packages. Although all of these types of "software" are ultimately translated into the "machine language" instructions which the 6502 can understand, a good purchased program works hard to hide the inner workings of the Apple from the user. Even if you choose to write your own programs in BASIC, Pascal, or some other language, the operation of the 6502 is handled for you, automatically, by the internals of the programming language.

In the Apple II and II+, there are nearly two dozen chips directly having to do with memory. In the //e, most of this has been reduced to eight chips arranged in a row along the front (16 along the right side in the //c), a second 40 pin DIP (the Memory Management Unit or MMU for short) and one or two more devoted in part to a special permanent program called the Apple Monitor. This Monitor is not the TV-like object you see things on, but is a "logical" monitor—it permits a human to peer into the inner workings of the 6502.

The Monitor: Who Speaks for the Apple?

If the Apple can be said to have a personality of its own, then that personality is written in the machine language computer program etched into the silicon memory chips in which the Monitor resides. Every time the Apple wakes up, the first thing that stirs is the Monitor. It is "Big Brother" to the chips of the motherboard, conscience to the 6502, and mediator of the rhythm and nature of conversation between you and the computer. In substance, it is a collection of several dozen short, fast, efficient computer programs which are burned into permanent storage on every Apple ever shipped. The Monitor is an example of a whole genre of permanent, unerasable software-on-a-chip often called "firmware."

For example, one of the Monitor programs constantly and automatically checks whether or not a key has been pressed.

You turn on your Apple, stand back two or three feet and just stare at it—meanwhile, having no way of knowing how detached you really are, and deadly intent to never, ever, ever, miss a keypress through inattention, the Monitor keeps checking the keyboard again and again and again, thousands of times a second. When it detects a keypress, it hands over control to another Monitor program that figures out which key was pressed. Yet another program then checks a list somewhere to find out if the Apple is supposed to do something special when that key is pressed, and so on—you get the idea.

If you know a true computer hacker who actually owns an Apple II (not a //e or even a II+) you can watch the Monitor pop to life when the machine is turned on.

The Monitor is much more secretive in the II+, //e and //c. You only get to talk to it directly during major computer disasters. Instead, during normal times, from almost the instant you turn on the computer, the Monitor turns on an Interpreter which understands the "English-like" commands of the BASIC computer language and translates them into the 6502 machine language instructions the Monitor can understand. Better yet, it also tries to turn on your disk drive and read in an even more "user friendly" program which may actually understand real English. You type in a question, the program from disk translates your English question into a question in the BASIC computer language, the BASIC Interpreter translates it into a machine language request (perhaps involving the Monitor), and finally the 6502 understands and does something to find an answer.

The Monitor and the BASIC language Interpreter are stored in permanent memory on specialized memory chips on the motherboard. The Monitor and the BASIC interpreter are always there, even when the Apple is off. The chips in which they reside can be read by the 6502. However, they are closed books in the sense that nothing can alter their contents. They may be read, but they may not be written to. In their infinite wisdom, the computer people from Southern California have come to call this type of memory Read Only Memory. The short form is ROM (always pronounced like Tom). This is the first of the four famous computer buzzwords (Bits, Bytes, RAM and ROM) about which you will be hearing more later.

Usable Memory: Memories that Change

The Monitor is a wonderful thing, but it is unchanging. It would be awfully boring if the Apple always did exactly the same thing every time you used it; ROM is important, but it is not enough. The changeable memory of the Apple is in two very different forms. The first exists in chips on the motherboard—it is lightning fast, but it is expensive and takes up precious space. The second type of changeable memory is physically outside the computer on some sort of magnetic recording surface—usually a 5 1/4 inch floppy disk.

Unfortunately, the 6502 can't work on any program, text, data, etc., unless that information is actually in the DIP chip memory on the motherboard. This means that anything stored in the external disk memory has to be copied into the motherboard memory before it can be used. If you were working on some information in the computer, and then needed to use something that was stored on the disk, you'd have to wait anywhere from two to 30 seconds before you could continue with what you were doing—this all but defeats the whole idea of a fast working microprocessor. The mitigating advantage of disk storage is that it is very cheap in comparison. It still costs about $50 to buy enough DIP chip memory to store a 40 page document, but only about 25 cents to buy the same amount of storage on a modern dense floppy.

There is one more major difference between the changeable memory on disk and the changeable memory in DIP chips—"volatility." Memory in DIP chips evaporates every time the power is turned off—it is absolutely gone for good in an instant and can never be retrieved. Memory on a disk, however, is exactly the same as music on a standard tape recorder tape—you have to make a fairly determined effort to erase it. Motherboard memory is fast but evanescent; disk memory is slow but lasting.

The final important conceptual difference between these two types of changeable memory is that information on disks usually has to be read in fairly long streams. If you wanted to see whether a sentence stored on the disk ended with a period, you'd probably have to read in the whole sentence starting from the beginning. With DIP chip motherboard memory, however, the 6502 can always look at the exact single smallest unit of information it is interested in. It does not matter where in motherboard memory it is stored, all locations are equal in the eyes of the 6502. This free ability to access all of this sort of memory at full speed has been called "random access"—and, for those of you who haven't figured it out already, the appropriate buzzword for describing fast, changeable, motherboard chip memory is Random Access Memory (RAM, the second of the four famous buzzwords).

Electronic Basis of Computer Memory: The Measure of Memory

Perhaps the biggest difference between memory in the human mind and memory in a computer is that in the computer memory can be measured in neat little units. The nature of these units grows out of the heart of hearts of microelectronics and its implications extend upward to the highest level of computer function. In this measure lies the link between your thoughts and the flow of electrons deep within the computer.

This nearly metaphysical link is accomplished by one of the great and marvelously original inventions of this century. The device is simple in conception. It is no more than an individual on/off switch—distinguished because it can be thrown nearly instantaneously, has no moving parts, and can be fabricated through photolithography in sizes as small as the nucleus of a single human cell. The switches are called transistors. There's more about this sort of thing in a later chapter, but for now the point is that two states, on or off, underlie all of computer electronics. The rest of electronics, in fact the rest of the universe, doesn't usually work this way.

Normally, you must allow for infinite variation between any two absolute points. In the mid 1940s however, computer designers abandoned the "linear" variation common to most electronic devices of that time in favor of discrete, on/off electronics, now called "digital" electronics. This choice was actually made before transistors were invented, and the first electronic computers used a kind of electro-mechanical switch in which a piece of metal actually had to swing across the switch box with every change. With the invention of transistors, it became possible to throw a switch far faster than a neuron in the brain can respond to a stimulus.

Logic and Memory from Transistors

In computers, these switches are arranged in groups that have "logical" decision-making capabilities. In one typical arrangement, two "wires" lead into a group of transistors called an "AND gate," and one "wire" leads out of the group. IF the first line coming in is coming from an "on" transistor group elsewhere AND the second line coming in is also coming from an "on" transistor group elsewhere, THEN the "AND gate" will turn its output line "on." This clump of silicon has just checked out some information from its environment, evaluated it, taken an action based on the information, and announced it's decision to any other groups which happen to be listening (by being attached to its output line).

Clumps of these "logic gates" can be connected together to form a memory group called a "flip-flop." These flip-flops usually involve about 20 or 30 transistors assembled into three or four logic gates. The gates are linked in a sort of electronic figure eight which you can learn more

about later, but the most important thing is that once a flip-flop is either set to the on condition or to the off condition, it will stay that way even after the incoming signals are removed as long as you don't turn off the computer. A transistor or a logic gate alone forgets what it was doing the instant you stop talking to it.

Some microcomputers use these flip-flops for all of their changeable motherboard memory, but in the Apple only the 6502 itself, and a few isolated chunks of special memory here and there on the motherboard, use flip-flops. When the first Apple was being designed, most computer manufacturers were choosing to use only flip-flop based memory, but Apple chose to use "dynamic storage cells" instead. At the time, the choice was probably due to flip-flop memory being far more expensive and requiring much more space than was available on the Apple motherboard.

Dynamic Memory

Dynamic storage cells are similar to flip-flops in conception, but they borrow from the old linear electronics, mix the old with a bit of very modern digital electronic trickery, and come up with an interesting sort of hybrid. The good news is that each cell requires the equivalent of only two transistors, but the bad news is that they spend a great deal of their time somewhere between on and off.

This irresolute approach to reality is all very scandalous from the point of view of the digital circuitry that surrounds them, and, as a result, just about every other chip in the Apple invests a great deal of time in a process of reminding the dynamic storage cells to please get themselves either completely on or completely off. If it's all done correctly, you never have to actually repeat to a cell what it is supposed to contain, you just have to keep on telling all of them to be tidy about it.

This process is called "refreshing" the memory. The whole business certainly sounds a bit cockamamie, but it is reassuring to know that as the years have gone by virtually all microcomputers have come to use dynamic storage cells rather than flip-flops. The initial decision sharply reduced the cost of Apple memory and played an important part in the Apple's early success.

These storage cells are designed so that the 6502 can address each individual one in two different ways. The 6502 can choose simply to "read" the storage cell by looking at the output line without changing the internal state, or it can cause the storage cell to flip (i.e., from on to off or vice versa). This is called a "write." The official buzzword name for the quantum of information embodied in the on or off state of one single storage cell is the "bit." One bit of information is the smallest amount that can be stored, changed, or read, and the information that can be contained in one single bit is limited to a seemingly unambitious two states: 0 or 1. And let's be frank, one bit just isn't that informative.

Bits and Bytes

Now, suppose you take two of these storage cells side by side. Both could hold a 0, or the first could hold a 1 and the second hold 0, or the first could hold a 0 and the second hold a 1, or finally both could hold 1s. With two storage cells, then, you get four possible states. In fact, you could store any number from zero to three as a pattern of 1s and 0s in these two cells. Three cells get you any number from zero to eight, four cells get you any number from zero

to 15, ad infinitum. If the 6502 wanted to know what coded number was stored in this group of four storage cells, it could go through one by one and read them in order. There is an alternative though. A great deal of time could be saved if there was a single address for the whole group. The 6502 would specify a group of four storage cells, and the group would automatically announce its collective status. After all, the 6502 doesn't really care about the value of each bit in and of itself, rather it wants to know the single value encoded by the whole group.

Aha, but is four the most efficient number of cells to put in a group? If you had a group of 10 you could get a number as large as 1,024 in a single read (that's two times two times two, done 10 times). In fact, large mainframe computers which specialize in number crunching tend to have as many as 64 storage cells in every group, so a single read gets them the potential of tossing around an inordinately large number such as the number of inches from here to the sun or some such. Since few Apple users need to operate with such large numbers on a day-to-day basis, a great many Apple storage cells would go unused most of the time—and remember, motherboard memory is expensive.

Eight Bits for ASCII

The number of storage cells grouped together in the Apple probably was determined more by letters than by numbers per se. A strong suspect for being the responsible party is the old teletype machine. A teletype is usually a 1950s or 60s vintage typewriter-like device which uses a bizarre spinning wheel to convert key presses at its keyboard into a series of digital on/off signals which are then sent over the phone lines (see Figure 1.3). A code was developed which had enough options for all capital letters, small letters, numbers, asterisks, parentheses, etc., as well as for a number of "control characters" used to warn the receiver about any special formatting of the incoming text.

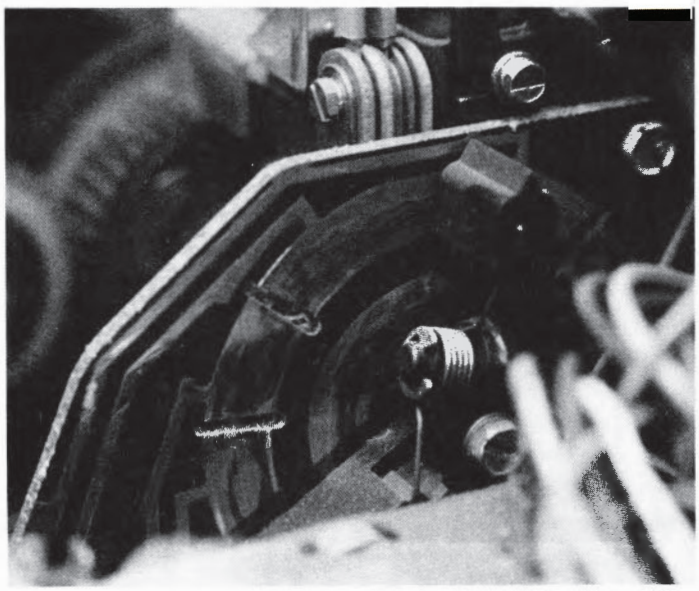


Fig. 1.3 Old Teletype mechanical communication device (circa 1960). The bits of the ASCII code are placed at different positions around the wheel. As the brush spins, the bits are picked up and transmitted "serially," one after another.

The code allowed for 128 different signals, so each character could be represented as a pattern of seven bits (two raised to the seventh power equals 128). However, since the object was to transmit the information over the phone lines, the designers of the code added one extra bit to each character pattern which they then used in a very clever way to let the receiver know for sure if static had garbled the transmission.

The code for each character is a set of ones and zeros (i.e., "010 0111"). This particular coded character has four ones in it. Whenever a code had an odd number of ones, however (i.e., "011 1000"), then the machine would automatically put an extra one on the end of the code ("1011 1000"). In this fashion, every code received must have had an even number of ones at the time it was sent. On receipt, each code was checked for this "even parity" and then this unneeded eighth bit was stripped off and thrown away.

The name for this code hasn't quite achieved buzzword status, but you do hear about it quite a bit. It is "ASCII" (pronounced "as key") which stands for American Standard Code for Information Interchange.

Any computer which might have to communicate with some other computer-like device was going to have to use ASCII, and so eight bits was established as the practical minimum for designing microcomputers. The buzzword term for eight bits is a "byte." You can buy a plug-in card for your Apple which handles numbers in 16 bit "words" or even in 32 bit "longwords," but eight bit bytes are the lowest common denominator. Only eight bits can be used at a time when you are tossing around ASCII alphanumeric characters in applications such as word processing or communications, so more brawny processors can actually be less efficient for many computer uses.

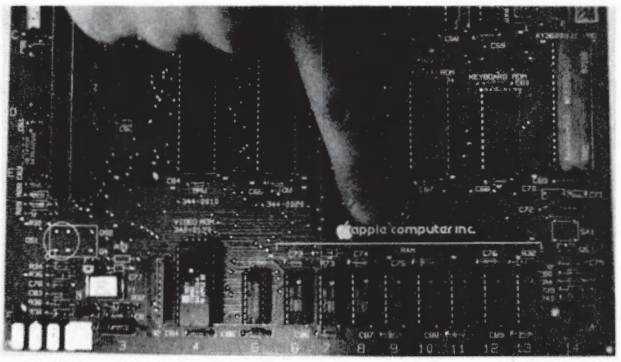


Fig. 1.4a (above) A row of eight RAM chips in the //e.

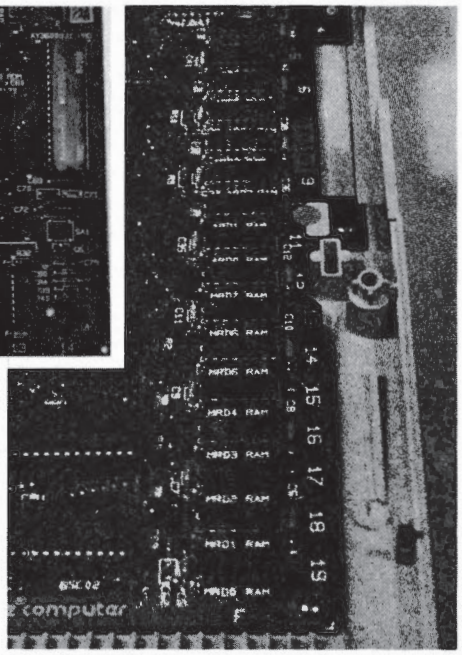


Fig. 1.4b (right) A row of 16 RAM chips in the //c.

There is one last surprise about the grouping of the eight storage cells into a single addressable byte. If you look back into the Apple at the motherboard and locate the RAM chips, you'll notice that the chips are laid out in groups of eight (see Figure 1.4). Each byte actually has only one of its eight storage cells in each chip. The 6502 asks for something like the front left bit. Then all eight RAM chips in the row each simultaneously announce the contents of the one bit they have in that position. One "wire" leads from each chip in the row to one of eight pins on the 6502 DIP package. The 6502 looks at the eight pins and sees an entire byte all at once.

In the Apple II and II+, each chip has about 16,000 bits on it so a row of eight of these chips can store about 16,000 bytes. In the //e and //c, each memory chip has 64,000 dynamic storage cells on board, so that a single row of eight chips holds 64,000 bytes. The standard abbreviation for 1,000 in computer lingo is the letter "K." Alright then, are you ready? The next time you say that your Apple //e has 64K bytes of RAM, you're going to know exactly what you're talking about in a sublime and exact sense.

Organization and Limits of the Memory: Forwarding Address

Ah yes, you say, but why this pervasive number of 64K? Why not 70K or 197K? A completely accurate description of exactly how much memory the Apple can use turns out to be surprisingly technical and complex. The details are spelled out precisely later; for now though, let's assume it can actually use exactly 64K and talk about why.

At the very beginning of the discussion of memory we said that the most important memory was actually on the 6502 chip itself. Some of this memory is like ROM—it stores permanent internal instructions called microcode. The volatile part of the memory can store a total of eight bytes. Each of these eight bytes has a unique function and an individual name. They are called microprocessor "registers" and are used in the actual execution of machine language instructions by the 6502. Three of these registers are used as scratchpads for fairly general purpose temporary storage, but the remaining five are responsible for remembering the location of the program the 6502 is currently executing, and for guiding the 6502 through the execution of the instructions.

Every 6502 instruction is at least one byte in length, some are two bytes long, and the longest take three whole bytes to make their point. An example of a three byte instruction is as follows: 1)"I'm about to give you the address of a number you'll be needing," 2)"This byte is first half of the address," 3)"This byte is the last half of the address." What just happened was that a program handed the 6502 a memory address which was two bytes long. This is the longest address that it's possible to give to the 6502. Two bytes is 16 bits. A 16 bit address can be any number from 0 to 65,535 (2 to the 16th), but no more.

This is the proximate limit on the amount of RAM the 6502 can conveniently speak to; it is the origin of the famous 64K. When the 6502 was designed in the mid 1970s this seemed like more than enough memory locations, and no one gave serious consideration to the use of four byte instructions or of putting in a 24 bit register to hold a three byte address.

It happens that there is some trickery which can be done to get around this limit, which is why the //c can have 128K, and why several companies sell large memory expansion boards for the Apple II, II+ and //e. However, this trickery is not convenient, and programmers have

been slow to go about taking advantage of it. Most of the popular commercial programs you can buy were written at a time when RAM memory was much more expensive than it is now and many Apple owners had only a total of 48K installed in their machines. The programs allowed you to add memory up to the full "natural" amount of 64K but did not make allowances for the trickery to expand further. Some Apple owners who are very sophisticated in writing their own programs do use much larger ranges of memory, and it is now possible to use large banks of memory with MagiCalc or VisiCalc, but for the most part we'll all just have to wait for the bulk of commercial software to be upgraded.

Floppy Disks and Disk Drives: Spinning Memory

Disks and disk drives got some mention earlier as inexpensive but relatively slow changeable memory; but these are fascinating devices in their own right and require substantial care and feeding. The fundamental logic behind the design of computer disk memory flows from a point already made. The 6502 can't read bytes of information from the disk directly. To use information stored on a disk, the 6502 must go through a two step process. First, it has to cause a chunk of information to be copied into RAM, and only then can it begin to use the bytes of information. Once the data is copied into RAM, the disk drive is simply turned off.

This two step process often causes some confusion for first time users who are thinking of a phonograph record or a tape recorder. If you take the record out of the record player, there's no more music. Not so with computer disks.

The first crucial variable in the design of the disk system is the size of the smallest chunk to be read by a single command. Although information moves from the disk to the Apple at a high rate of speed once the information has been located, it may take more than a second to get to that step. First, a command must be sent to turn on the drive's motor to get the disk spinning. The red light on the drive turns on immediately, but the Apple still has to allow time for the disk to go from 0 to 300 revolutions per second. Then the reading mechanism has to be physically positioned over the correct location on the disk. If this work were done one byte at a time, it could take nearly a minute to copy in one 40 column line of text (in a worst case scenario). Apple has settled on a chunk size of 256 bytes at a time. When the Apple knows exactly which chunks it wants and where they are on the disk, it's able to read in about 1000 bytes a second in a steady stream—this means it takes about two seconds to fill up the equivalent of a full 80 column by 25 line screen when reading directly from disk. These 256 byte chunks on the disk are called "sectors."

On an Apple diskette the sectors are laid out in 35 concentric rings, not in a spiral. Each ring or "track" is broken into 16 sections, each of which is a "sector." Therefore there are 35 times 16 equals 560 sectors. Since each one contains 256 bytes, there's room for about 143K bytes on one Apple diskette. The Apple uses about 15K for its own purposes and you get to use 128K of storage on one Apple diskette. Most word processing programs keep at least two copies of your text on each diskette, so for practical purposes this means you're limited to about 30 or 40 pages of text on one diskette. There is no substantive reason for this 128K limit except that it reflected the state of disk drive technology several years ago when the system was set up. The newest high-tech Apple-compatible drives have 154 tracks each with 30 sectors on one side of a 5 1/4 inch diskette.

The Disk Operating System: Monitoring the Spin

If disk space is so limited, what is it that the Apple does with that missing 15K? You'll no doubt recall the system Monitor programs which we said gave the Apple its personality. That Monitor was written before 5 1/4 inch disk drives were commercially available (early Apple owners used a cassette tape recorder for external storage), so it doesn't include the programs required to operate the disk drive and to keep tabs on what information is stored in which sector. The machine language programs which supervise the management of the disk drive are called collectively the Disk Operating System (DOS). You get one copy of the Apple disk operating system when you buy your Apple.

This set of DOS programs has to be copied into RAM memory every time you turn on the Apple and want to use the disk drive. The one advantage of distributing these programs on disks is that it is comparatively easy for Apple to distribute improved versions of the DOS programs after your computer leaves the showroom. Older Apples use version 3.2, most current Apples use 3.3. The newest disk operating system for the Apple II is called ProDOS. It is completely new and is far more powerful, versatile and easy to use than DOS 3.3.

Loading DOS into RAM: Footware for Memory

Some of you may have noticed a logical inconsistency in the previous paragraph. If DOS has to be in memory before the disk drive can be used, then how does the 6502 know how to copy in the DOS itself? This problem is solved by a trick called "bootstrap loading" (implying that DOS pulls itself into memory by pulling on its own bootstraps) and typically referred to as "booting" the diskette. One of the chips on the disk drive controller card is actually a small ROM chip. It stores a tiny little "mini-DOS" which the 6502 can read and execute quite easily.

To turn control of the 6502 over to this mini-DOS, Apple II owners had to type commands into the Monitor in something resembling machine language. The Apple II+ was sold with two new ways of booting: the "PR#6" command in BASIC, and an "Autostart" version of the main Monitor which automatically started executing the mini-DOS whenever the Apple was first turned on. In the //e or //c, you can boot the disk drives by hitting the Control – Open-Apple – Reset combination on the keyboard or by using any of the previously available methods.

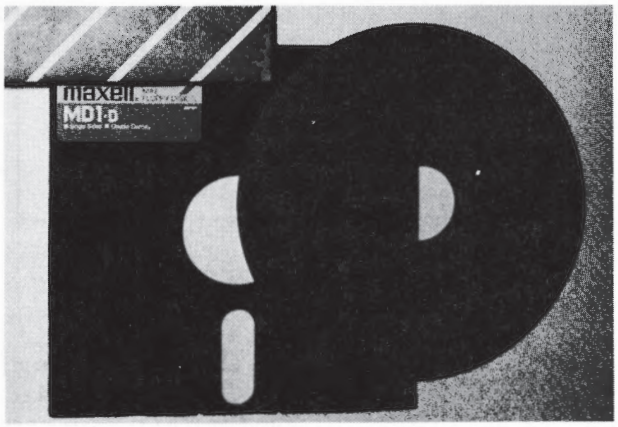


Fig. 1.5 The inside of a floppy disk.

When this mini-DOS program is run, it copies in some specific sectors and turns control of the 6502 over to the short program it has just copied in. This program is nothing but a set of instructions for reading a larger bunch of sectors, and so on, until all of the 10K of DOS is copied into RAM. The image of DOS is then moved to the higher address numbers in memory and the lower address part of memory is left available for other programs and data to use. DOS only has to be loaded once when the computer is first turned on. However, since it stays in RAM memory, it is destroyed whenever the Apple is turned off and you have to be sure a copy is available on the first diskette you use when the computer is next turned on.

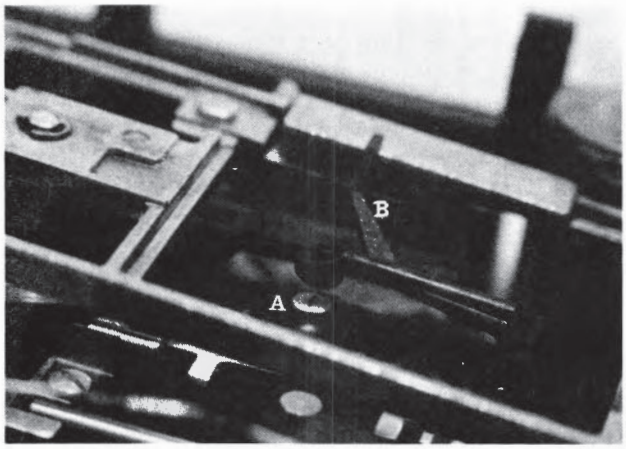


Fig. 1.6a Inside the Disk II drive. The small white circle with the black line is the read/write head (A). When you open the drive door, you lift the metal frame and pull the black pressure arm (B) up off the head.

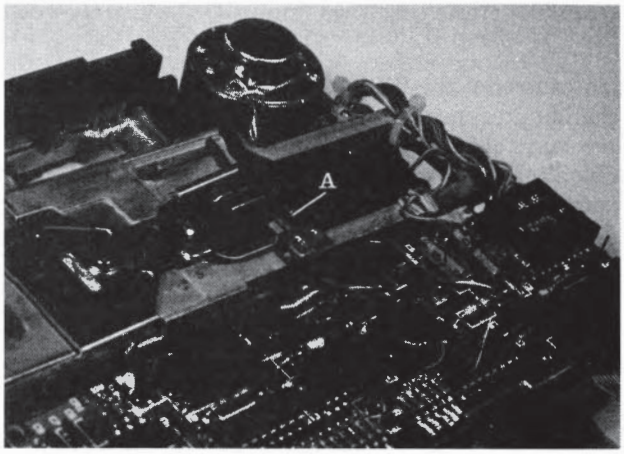


Fig. 1.6b Inside the Disk //c drive. The small square with a black line is the read/write head (A).

Initializing a Diskette: The First Spin

A brand new diskette is a clean smooth magnetic slate (see Figure 1.5). It has no tracks and no sectors. Normally, DOS is completely unable to do anything with such a diskette, and it certainly can't be booted. To use a new diskette you have to first bootstrap-load a good copy

of DOS from some other diskette. Once DOS is loaded, you can use one of the DOS programs you have just loaded to prepare the new diskette for use. The command you type is usually INIT HELLO and this calls up the initialization program. The process of initialization involves putting little magnetic marks in each site on the diskette which will later hold a bit of information. If you stopped the process at this point, clipped off the diskette jacket and sprinkled iron filings on the diskette, you could actually see the tracks and sectors (this is OK for laughs once or twice, but it is an expensive trick).

There is a single read/write head in the disk drive which sits beneath the spinning disk (see Figure 1.6). Directly above the head is a little swinging arm which presses the disk surface down onto the head. The read/write head and the arm can be moved from the outside of the disk to the inside in 35 little steps, all under command from the 6502. If you listen carefully during initialization you will first hear a loud buzz when the head is yanked all the way to the outer track; next you'll hear 35 quiet little thumps as the head steps inward to write each successive track. When you open the drive door, you manually lift the little arm off the disk and thus bring the diskette out of contact with the head.

Once each sector has been marked out, the program goes on to write an address in the first part of each sector. When this process is completed, it becomes possible for the more standard parts of the DOS program to copy from or write to any individual sector out of the 560 which it so chooses. Before finishing, a complete copy of DOS is written on the first three tracks (starting from the outside, working in), and a filing system is set up on track 17 (halfway in from the outside). Finally, one extra program in Applesoft BASIC called "HELLO" is copied onto the diskette. Formatting from a ProDOS menu is similar, but ProDOS itself is only copied onto the disk if you decide you want to put it there for the purpose of creating a new startup disk.

The Directory: Keeping Track of Tracks

The filing system is the part of DOS you're forced to be aware of the most often. Although information on the disk is actually arranged in physical sectors, most users would like to refer to an entire coherent collection of information by a single name. The term for such a collection is a "file." A file might contain an entire word processing program in either BASIC or in machine language, or it might contain a letter or report you're working on.

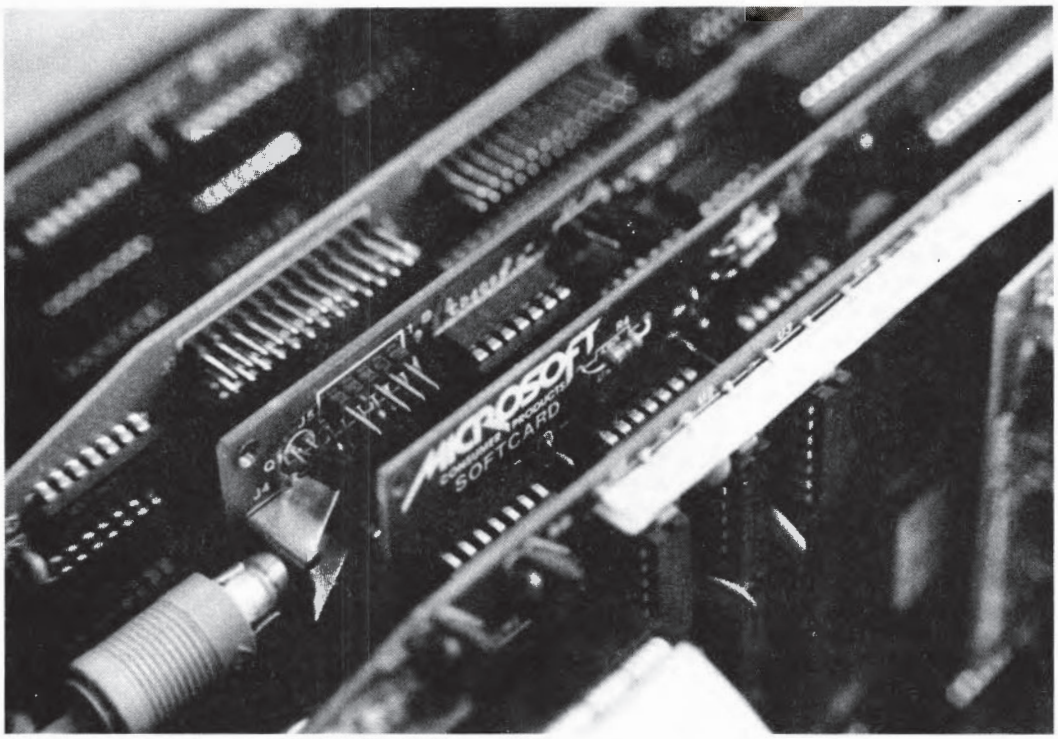
Let's say you're using a word processing program and you have typed a three page letter into RAM memory. When you're finished, you usually tell the word processing program to save your text by copying it out onto the disk as a file called, for example, MYTEXT. The word processing program tells DOS the name you chose, and it tells DOS where in RAM it can find your text. DOS then copies your text, 256 characters at a time, onto a series of sectors on the disk.

DOS is not required to use the sectors in order. It could put the first bunch of characters in sector 4 of track 19, then the next bunch in sector 10 of track 21, etc. A free wheeling system like this permits DOS to make the most efficient use of your disk as you create and remove files. However, while it's bouncing around out there with your text, it has to keep a very careful record of which sectors it has used and in what order it used them.

DOS begins by writing the name of your file on the filing system track of the disk, then it keeps a list of tracks and sectors as it uses them. This list is called the diskette "directory." Later on, when you want DOS to retrieve MYTEXT, it checks the directory to see if there is

a file by that name. When it finds it, DOS is then able to locate the list of sectors. It reads them in the appropriate order, copying the contents of each of them into RAM. When it is all done, you get back the complete text you started with, ready for further work. In the body of your text there is no remaining evidence of the partitioning of the file into sectors. All you are aware of is your text and the name of your file.

This Directory is a potential Achilles heel. If it gets damaged, you may be unable to retrieve your text even though the text itself is completely intact. There are wizards out there who specialize in retrieving data from diskettes with damaged directories, but the best cure is prevention and the only means of prevention is to make duplicate copies of important files before disaster (which, by the way, is inevitable) strikes.


MICROSOFT
SOFTCARD

Chapter 2

Output, Input and Expansion

Characters on the Screen: What You See and What You Get

Once you've closed the top of the Apple, hooked up all the cables, turned it all on, and let the disk drive do its thing, the two most engaging aspects of the Apple are the keyboard and the screen. The clear and obvious mental analogy for just about everyone is the trusty old typewriter. In a typewriter, the words on the paper are the be all and end all of what the machine and your task are all about. Not so in a computer, not so at all. In fact, from the 6502's point of view, the pattern of light and dark you see on the screen as words is a lot of meaningless gibberish about which it would just as soon forget.

In fact, relieved of the burden of dealing with humans, the Apple could do all of its work without ever drawing a picture of what it has done. The characters you see and the way they are displayed are entirely an afterthought for the computer. Fortunately, computer programmers are typically humans, (user compatible warmware?) and most of nearly every computer program ever written is devoted to managing the appearance of characters on the screen.

Screen Memory: Holding On to the Cast Off Characters

When the 6502's work results in the production of numbers or strings of alphabetic characters which it believes the user might care about, it starts blindly dumping the codes for these numbers and letters into a RAM storage area (computer folks always call these RAM storage areas "buffers"). There is one area in the Apple's RAM which is always used in an agreed upon way for representing screen information. It is conveniently thought of as a grid with 40 by 24 equals 960 pigeon holes.

One of the Monitor programs catches the character codes as they come out of the 6502. Then, handling them one by one, it figures out where to put them among the memory locations which represent the 960 spots in the 40 columns by 24 lines available.

It sounds like an extremely simple task, but it is not completely intuitive. One typical pattern is to start at the bottom of "screen memory" on the 24th line and the first column. As new characters turn up, the program steps to the right one column at a time until it hits the right margin. When the next character comes in, the line that was just finished is copied over again into the 40 memory locations for line 23, and everything that was in line 24 is erased. On the screen this looks like the bottom line has just rolled upwards—a process called "scrolling"—but, as you see, the actual details aren't quite that elegant.

Once the whole screen is full (when the 961st character arrives), the top line appears to get rolled up just above the top of the screen. In fact, it has fallen off the edge and has quite completely disappeared from screen memory. That doesn't mean it's gone altogether. Remember, the screen display is quite incidental to what the 6502 is actually up to. If you ever want to see that line again, the screen management program which has been doing so well up to now is completely unable to help. Some other program is going to have to know where those character codes actually are on the disk drive or elsewhere in RAM and then bring them back to the screen program.

80 Column Display: Improving the View

If you have a //e or //c, you're probably wondering why there is all this harping about 40 columns. If you have a II or II+ you know why and you have probably already shelled out an extra two or three hundred dollars to escape the tyranny of 40 columns. If it's so unpopular, why did Apple choose 40 columns in the first place?

You may remember an article in TIME magazine in which one of their writers complained about the fact that computer manufacturers call their screens "monitors" when everyone knew they were really just TVs. Not true. There is an important difference. Probably the least insightful thing the inventors of the Apple decided was that few people would buy a special monitor for their Apple, and that hook-ins to home TVs would be the order of the day.

Early Apples were sometimes shipped with an attachment that turned the normal signals into radio-frequency information a TV could understand, and you get one of these RF modulators free with a //c. But if alphabetic characters were going to be readable on a TV screen, they were going to have to be relatively big characters. As it happens, 40 columns by 24 lines of capital letters is all your average TV can handle. And thus early Apples came by their humbling deficiencies of no lowercase letters and only 40 columns. This wasn't just a mechanical thing. The limitation was written into the heart of hearts of the Monitor program and even into the design of the BASIC computer language interpreter. Even in the //e you have to use machine language-like commands to supplement Applesoft BASIC to get control of any column after the fortieth. It's only in the //c that 80 columns and lowercase characters have been completely integrated into all levels of the system.

The other available standard for how many columns Apple might have chosen comes from another relic of early, mostly bygone computer days—the old IBM punch card. These cards and the great big card punching machines which are the great grandparents of modern CRT terminals allowed for 80 little holes to be punched across the surface of each card. Whenever an Apple owner wanted to use the machine as a way of talking to the huge mainframe computer, he or she discovered that the mainframe expected to be spoken to and heard from strictly in 80 column lines. Hence the advent of "80 column video boards" for the Apple from such companies as Videx and ALS.

For the most part, these cards don't use the Apple's screen memory grid, and they ignore what the Monitor screen management programs are doing. All that is required is to get a look at the character codes as they come out of the 6502 and to put them into the 80-column board's own built-in screen management system. These boards have their own RAM for storing the characters in an 80 by 25 display grid.

In the //e and //c an even more bizarre approach is used. Apple stuck with exactly the same 40 by 24 screen memory locations it had always used, but then it used one of those bits of memory address trickery mentioned earlier for going beyond 64K. Out in the //e special slot

(when even the simplest //e text card is installed), there is an extra 1K of RAM which thinks it has the exact same addresses as the normal screen memory. Half of the characters go to their normal location on the motherboard and the other half are placed in RAM out on the text card. The motherboard gets the odd numbered columns and the text card gets the even ones. In the //c, characters for odd numbered columns go in the eight RAM chips located towards the front of the machine and even ones go to the back (see Figure 1.4b).

Character Formation: Electron Calligraphy

All of the screen character handling talked about so far has led to a bunch of character codes being laid out in RAM. This sort of thing is nothing you can actually see. The final step is for another part of the Apple called the Video Display Generator to keep on scanning the lines of screen memory about 60 times a second and then to translate the series of character codes into a completely different kind of signal which operates the electron beam in your actual Cathode Ray Tube (CRT) monitor. The beam sweeps horizontally across the screen you're looking at and blinks on and off as it passes.

This is where the actual quality of the characters you look at comes in. If the beam swept horizontally across the screen a full 20 times before it got from the top to the bottom of each line of text that it was drawing out, then it would have the opportunity to render each letter as a beautiful and precisely formed shape. However, before it got to the bottom of the entire screen it would have had to make an awful lot of fast, precise sweeps. It turns out that seven passes for each line of text (not including the spaces between lines) makes a pretty good looking alphabetic character. Various 80 column-video character generator boards for the II and II+ use from seven up to 11 sweeps ($100 for the one that makes seven sweeps and $275 for the one that does 11), and the //e and //c use seven sweeps. In 1983, the Videx Corporation put Apple to shame and substantially upped the ante by releasing a plug-in Apple video board which generates 128 columns by 32 lines (or 160 by 25 or 80 by 48) all with 12 sweeps on each character line (this one lists for $380).

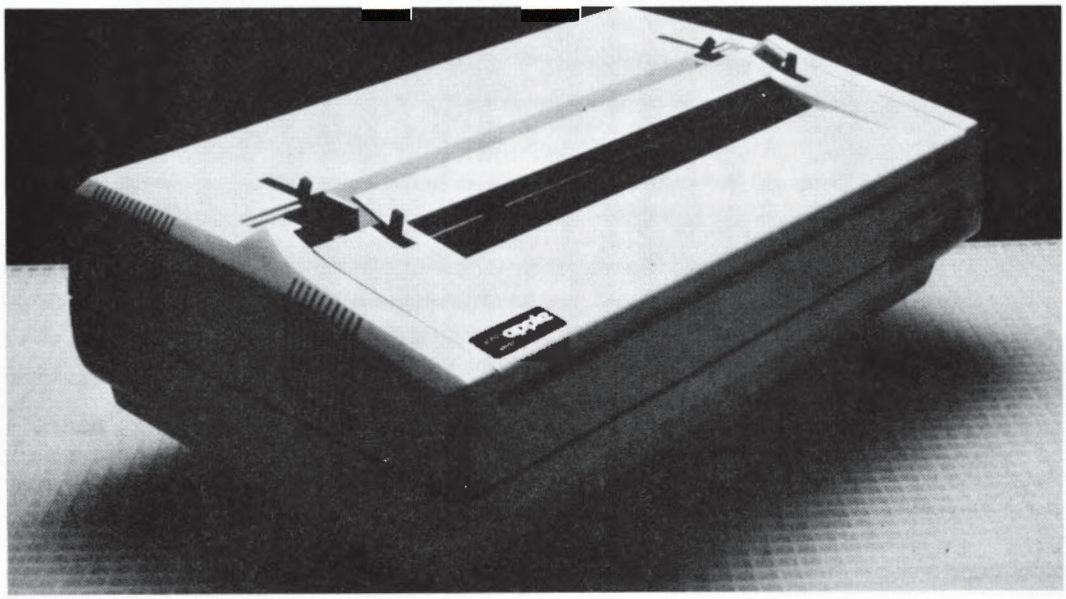


Fig. 2.1 Apple letter-quality printer.

Characters on Paper: Ink and the Apple

The operation of a printer that actually types on paper is logically identical to putting characters on the screen. The programs that arrange where letters will appear on paper are extremely similar, and in some cases identical to, the ones that arrange where letters will appear on the CRT screen. Of course, once the lines of text or numbers are appropriately laid out in RAM, the character generation step has to be done a bit differently. In nearly all cases, entire letters are generated one at a time; this differs from the situation on the CRT screen where you first generate the top one-seventh of all the letters on a line, then the second one-seventh, etc.

Once the command to print a letter has been sent, the details of character generation are left to the actual printing machine. The design problem for computer printers is that they've got to take an actual physical hammer, swing it through some distance, mash it through a ribbon onto the paper, get it out of the way and then get the next character ready for the hammer. All this real world motion seems hideously slow from the point of view of your average electronic device.

The fastest approach that is in common use for Apples is to constantly change the shape of the surface of the hammer. This is what is done in the "dot matrix" printers. The hammer is actually a "print head" made up of a vertical row of movable wires with their cut ends facing the paper. Individual wires in the row are either left inactive or driven forward to strike the paper several times as the print head advances across the character (see Figure 2.2). The sequence of dot patterns is altered according to which character has been called for by the computer.

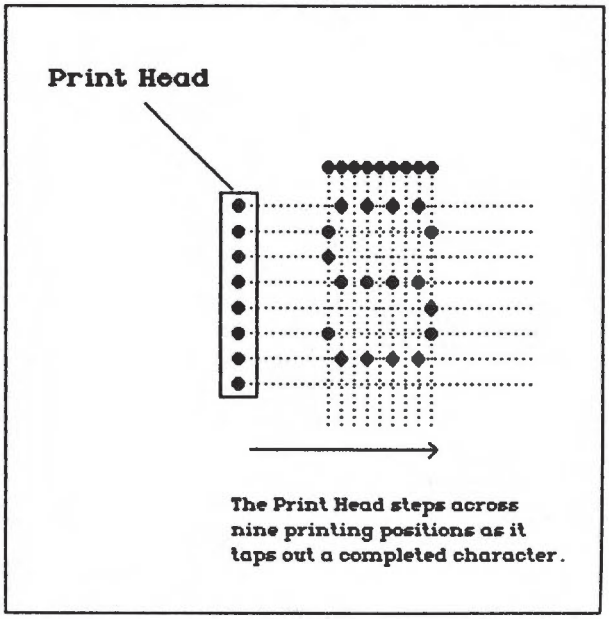


Fig. 2.2 The print head of many dot matrix printers is a vertical stack of nine pins. As the print head steps across the nine printing positions, it taps out a completed character. The sequence of dot patterns is altered according to which character has been called for by the computer.

A dot matrix printer which used a row of 20 wires with 20 horizontal steps across each character would produce a 20 by 20 matrix of 400 little dots for each beautifully formed character. However, each wire would have to be so thin that they might wear out a few days after you

bought your printer and it would take quite a long time to work across an entire line of text. As a result, the most rugged, high speed dot matrix printers tend to use a vertical array of nine wires with eight strikes across the character. Their characters aren't particularly pretty, but such printers can type as many as 200 characters per second and can last for years. Soon, however, dot matrix printers may be replaced by a new generation of printers which use a matrix of little ink jets instead of wires. This avoids the need for the print head to swing down onto the paper and also removes the durability limit so that very dense character matrices are possible.

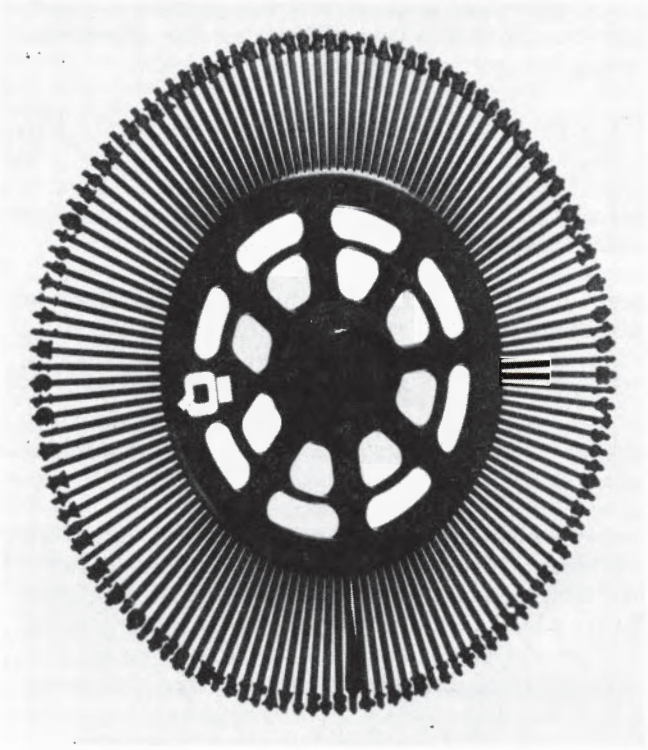


Fig. 2.3 The daisy wheel printer is an alternative to the dot matrix printer. Most daisy wheel printers use a wheel with 96 character strokes which produce typewriter-like characters. This is a 130 character wheel for a special Qume printer.

The alternate approach to printer character generation is a bit slower but makes perfect typewriter-like characters. The best version of this approach is called a "daisy wheel." The wheel has 128 petals (like spokes) each of which has a completely formed character on its end (see Figure 2.3). A given ASCII code is interpreted as an instruction to rotate the little wheel to the appropriate position to get the requested character under the hammer before it strikes. These "letter quality" printers also permit fancy formatting, such as superscripting and subscripting, by using other ASCII codes as instructions to roll the platten to slightly offset positions before striking in the character. The end result can be equivalent to the best expensive typewriter, but few of these machines can move faster than about 55 characters per second.

Keyboard Function: Fingerprints on the 6502

At one point earlier in the discussion about the physical distribution of bits in a byte of RAM you may recall a mention of eight "wires," one from each RAM chip in a row, leading to eight pins on the 6502 DIP chip package. The 6502 is able to look at these eight pins all at once

and see an entire byte of information. This byte could contain a machine language instruction, or it could contain a number from 0 to 255, or it could be the ASCII code for an alphanumeric character (letter or numeral). The 6502 looks to these eight pins (called "data pins") to find out almost everything it needs to know.

As it happens there's more than one way to put a byte onto those eight pins. Granted, the RAM uses them quite a bit, but the RAM does not have them all to itself. So who do you have to pay off in order to get an audience with the 6502's eight data pins? Actually, access is very freely available as long anyone who has something to announce follows appropriate protocol and goes through appropriate channels. The appropriate protocol is to have all eight bits ready and waiting to go, together and all at once when the 6502 asks to see them, and the appropriate channels boil down to two major routes and two relatively minor ones.

The first and most obvious route into the Apple is from the keyboard. When you press a key, a unique coded signal is generated, loaded into a cable, carried into the Apple, decoded into a one byte ASCII code and then held in a little buffer. Part of the Monitor program informs the 6502 that a keypress has been made and gently advises it to check the keyboard buffer to find out which key it was, as soon as is convenient.

In the eyes of the Apple, all of the keys are equal. None of them has any meaning at all unless some program it is executing has some special idea about them. Only the "Reset" key has a permanent special function, but even this can be modified by a clever machine language programmer who can turn it into a relatively innocuous key acting like any number or letter he or she so chooses. When you press a key, any key, you are only generating a code which must be thought about and kicked around a little in the Apple before it can cause anything to happen.

This is another aspect of computer function which can be confusing to a first time user accustomed to thinking about a typewriter where a key is a key is a key, and they all do the same thing every time you hit them. In a computer, however, to know the effect that a given keypress will have, you must know exactly which part of which program is running at the time you type it. For the most part you have absolutely no choice other than to teach your fingers a different keyboard language for each operating system you use (ProDOS, DOS, Pascal, CP/M) and for each different program.

Expansion Slots: Bytes from Different Directions

One striking feature of the II/II+ and //e motherboard that hasn't gotten any mention yet is that row of three inch long plastic slots along the back (see Figure 2.4). Each slot contains 50 metal contacts laid out in two rows of 25. The contacts in each slot are numbered from one to 50 beginning on the right side at the front end, going up to 25 at the back and working up to 50 at the front left. In each slot, the pins numbered 42 through 49 are connected directly to the 6502's eight data pins. There are literally hundreds of different kinds of electronic "cards" you can buy to stuff into these slots, but almost all of them live and breathe by placing bytes of information on pins 42 to 49 of the slot.

Once a byte arrives on pins 42 to 49 of any one of the slots, that byte is physically indistinguishable from a byte that is coming from the keyboard. Most of the time, the 6502 is able to keep track of where these bytes come from by limiting access to its data pins. No bytes are

ever actually put directly onto the data lines (which connect to the data pins). The keyboard or the slot cards always first put their bytes into their own personal little buffer next to the data lines. Next, they go about trying to tell the 6502 they're holding a byte which they think the 6502 should look at. Like any information manager surrounded by five or six sources of information screaming for immediate attention, the 6502 and the Monitor are forced to rely on a carefully laid out system of determining who has priority. Once a data source is chosen, the 6502 can send a signal which dumps the contents of the selected buffer onto its data lines. Only then can the 6502 find out exactly what the selected information source has to say. The //c has several "built-in cards" which act as if they were in slots, so the same basic scheme still applies.

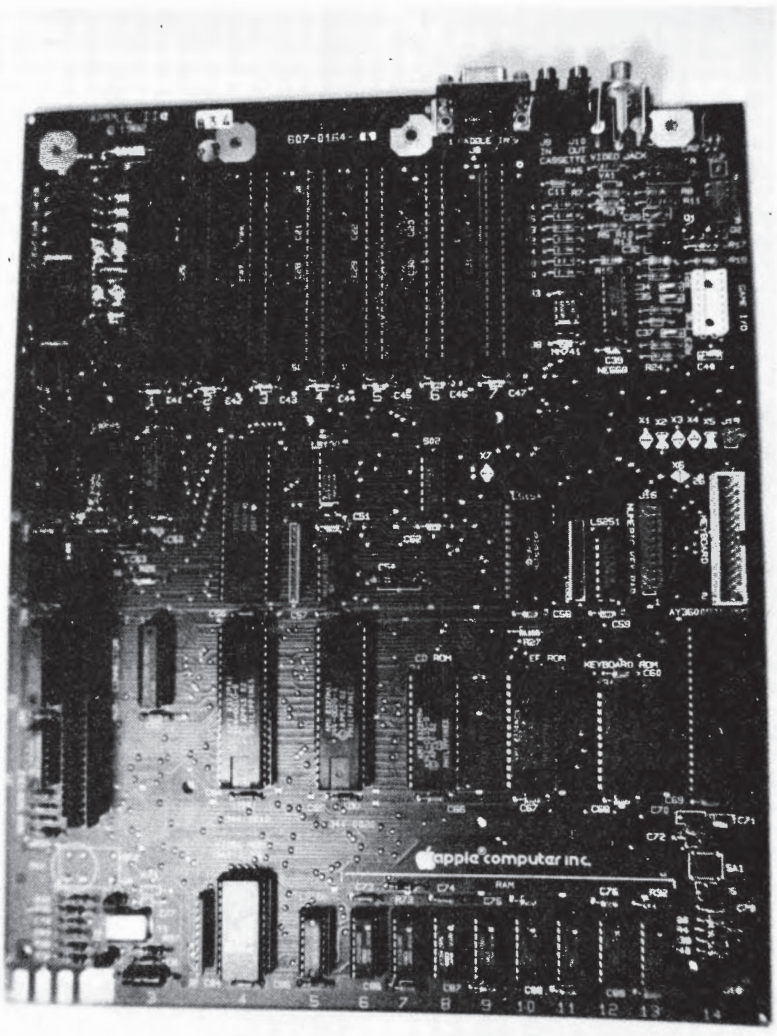


Fig. 2.4a The Apple //e has seven accessory slots and one auxiliary slot.

The source of the information coming in from the keyboard is obvious, but the various slot cards are set up to feed in information from a whole smorgasbord of kinds of sources. Almost everyone who owns an Apple has a disk drive controller card in slot number six. Among other things, that card collects in coded data from the surface of the disk and places it on the data lines. Another popular source is a graphics tablet. This is a board that you just draw on. As the special pen moves around on the tablet, bytes appear on the data lines which tell the Apple where the pen is. In this fashion, a coded picture can be loaded into the Apple's RAM and a special program can use the data to make a duplicate copy appear on the screen or to save a copy of the code for the drawing out onto the disk drive for later display or analysis.

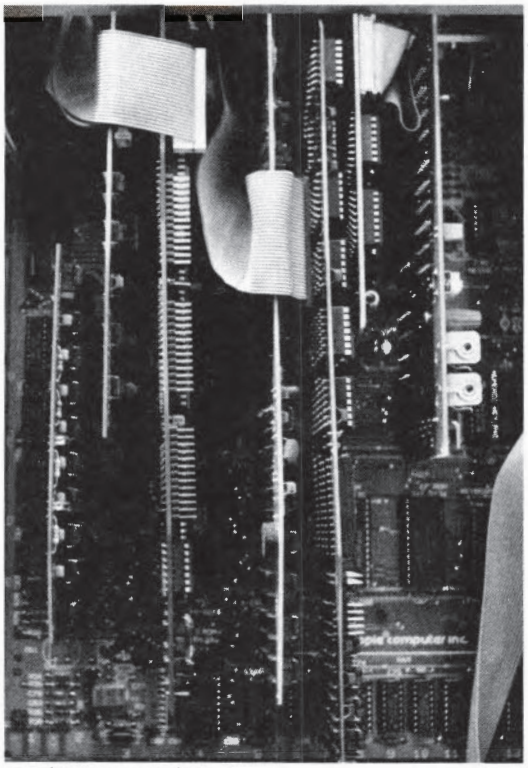


Fig. 2.4b (left) A fully stuffed Apple can be a dazzling machine.

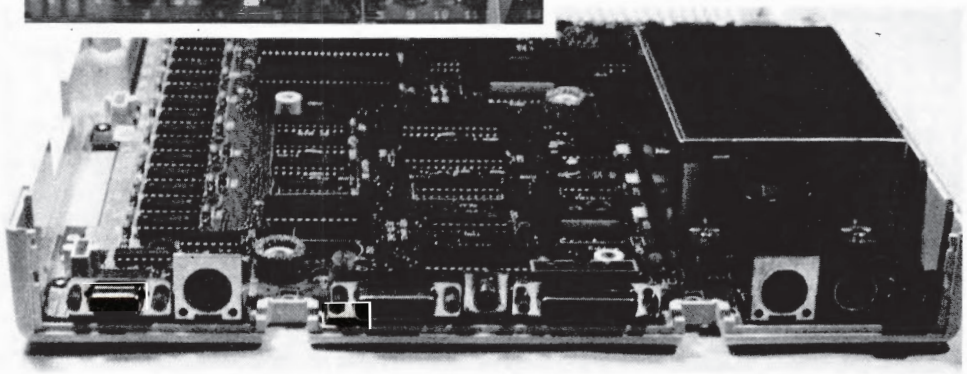


Fig. 2.4c (below) Behind all those connectors on the back of your //c is all the circuitry for two serial interface cards, an "80 column video" card, a mouse interface card, an "RGB color interface" card, and a disk drive controller interface card.

Serial Input: Getting a Word in Edgewise

After the keyboard and the disk drive, the most popular third source of information is from the telephone. For the most part, computers only speak on the phone to other computers. There's no reason why you can't build a device and write a program to let the Apple understand spoken English coming in on the telephone line, and, in fact, some devices which can do this in a limited way are available for the Apple. For now, though, most computer telephone communication is limited to the use of digital signals, usually grouped into standard ASCII code.

Earlier on it was noted that the required protocol for feeding bytes to the data pins demanded that all eight bits be ready and waiting to go. However, when ASCII codes are sent over the telephone lines this presents something of a problem. The whole world's phone system is based on information traveling in one direction having just one wire to travel in, yet you need to have eight wires addressed all at once inside the Apple. The digital on/off signals of the ASCII code are sent down the line one after another. The first bit arrives, then the second, and so on until sometime later (seemingly an eternity from the 6502's fast lane point of view) the eighth bit arrives. This pattern of arrival is called "serial," meaning that the bits come along in a linear series one behind the next. The way the 6502 wants its bits is lined up side by side in what are called "parallel" channels.

Fortunately, converting from serial to parallel isn't a big deal. The incoming signals are first converted from telephone tones into computer voltages by a "modem" device (to be examined in more detail later) and then the //c modem port or a slot card which engages in this sort of conversion just collects up the incoming bits and puts them into their proper place in a buffer side by side until all eight have arrived. Only then does the card make a request for the 6502 to read the contents of its buffer. By the time the 6502 actually does take a look, the serial bits have already been put in place and appear completely identical to a parallel byte.

CP/M and the Z-80: Two's Company

Applesoft BASIC, Apple DOS, and the 6502 are wonderful things that many people claim to love dearly. However, tens of thousands of Apple owners, do the great bulk of their computer work on a different computer. Don't be alarmed. Although this second computer is in many ways faster, more powerful, more versatile and more sophisticated, you get the whole thing on a single Apple slot plug-in board or //c expansion module that costs about $250. It uses the RAM you own, it uses the Apple's keyboard and screen, it operates the same disk drive and talks to the same printer.

When the first Apple was still a mere twinkle in Steve Wozniak's eye, two great microprocessors had already been born. One was made by Intel Corporation and they called it the 8080. A second had been developed by Motorola and it was called the 6800. A third company, MOS Technology, had looked over the design of the 6800, decided it was a great machine but that they could build it a little better and a little cheaper. Thus was born a close relative of the 6800—the 6502 of subsequent notoriety.

When Wozniak designed his first microcomputer he did it as a hobby project. He had no company and not too much money. At the time, an 8080 would have cost $50 or $60 and all the electronic parts distributors preferred payment via a company purchase order form. The 6502, on the other hand, which was available over the counter at hobby electronic stores, was much more reliable in simple machines, and was much cheaper.

This is not to imply the 6502 isn't a great microprocessor, but it may help explain some differences in the subsequent history of microcomputers based on the two. Many hobbyists and engineers found the 8080 more interesting because it was more elaborate, and programmers found that the same business market that was willing to pay extra money for an 8080 based machine was willing to pay extra money for more elaborate programs requiring larger amounts of expensive RAM memory. It is still the case that the fastest and most powerful programs for word processing and data management have been written in the language of the 8080.

It happened, however, that a couple of students at Harvard Business School were only able to afford an Apple when they wrote one of the best business program ever written for microcomputers—VisiCalc. For years, businesses that wished to use VisiCalc had no choice but to buy Apples. At this point Bill Gates comes into the picture. One of his major contributions to the history of microcomputers was to write a version of the BASIC computer language which could run on the 8080. He realized that he could sell copies of his version of BASIC and help make his language popular with a broader range of professional programmers if it could be run on Apples as well as on 8080 based computers. His company, Microsoft, produced the first "8080-based" plug-in computer for the Apple. With this card installed, an Apple owner had access to both all of the programs written for the 6502, as well as to the equally large number written for the 8080.

Unlike the 6502, the 8080 has never been associated with any one particular brand of computer. 6502 based machines like the Apple (and the Commodore) are usually quite insular and only run programs designed for one computer. On the other hand, 8080s have been designed into dozens and dozens of utterly different computers, but have always allowed nearly total "portability" of programs. A word processing program such as WordStar, which runs on one 8080 based computer, will almost certainly run on any other.

As mentioned earlier, the 6502 is really an update of the Motorola 6800, nonetheless 6502 programs do not run on 6800s and vice versa. In the 8080 world, however, "software portability" was at a premium and when newer versions of the 8080 were designed, the engineers made certain that any program which could run on an 8080 could also be "transported" onto the newer chips. Features were added, but nothing old was changed or taken away. Intel's second generation version is the 8085, but a different company, Zilog, produced far and away the most popular update of the 8080—the Z-80 (see Figure 2.5).

The souped up Z-80B processors, which are rapidly growing in popularity, run through program instructions at three times the speed of the 6502, but the language in which the programs are being executed is really the old and sage 8080 machine language. Some companies, such as Microsoft, have written programs in Z-80 language that won't run on 8080 or 8085 based machines. But since Microsoft sells it's own Z-80 card, they really don't mind the loss of portability—you have to buy their processor card if you want to run their programs.

Apple DOS and the Apple Monitor are written in 6502 machine language, so a Z-80 card sitting alone in an Apple can't talk to the keyboard, can't talk to the screen, and can't operate the disk drives—not a very productive state of affairs. The popular Microsoft Z-80 card doesn't even have any ROM. To operate the Z-80, you must have a Monitor program and a disk operating system written in 8080 machine language. When you buy a Z-80 for the Apple you are given a disk which carries both the 8080 machine language Monitor and disk operating system. The bootstrap procedure when the 8080 operating system is loaded starts out just like an Apple DOS boot, with the first steps being handled by the 6502.

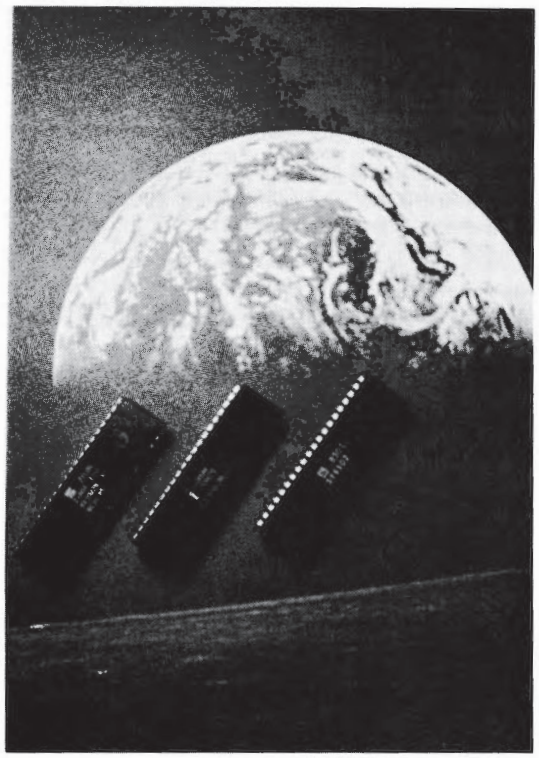


Fig. 2.5 Eight bit microprocessors are made by Zilog (Z-80), Intel (8085), and Synertek (6502).

At some point, however, when enough 8080 language has been copied into RAM, the 6502 does a remarkable little trick; it simultaneously turns itself off, turns on the Z-80, and tells the Z-80 the starting address of the 8080 machine language it has altruistically loaded. The Z-80 takes over, finishes the loading of the Monitor and disk operating system, and then starts sending messages to the screen that it has got control of the Apple and is ready to accept commands from the keyboard.

The Monitor and disk operating system used by 8080, 8085, and Z-80 based computers were first written in 1973 by Gary Kildall while he was an employee of Intel. He had a very early version 8080 microcomputer system and one of the first floppy disk drives ever made by Shugart. His operating system may not have been the absolute best possible one that could have been written, but it was first and it was very well done. The earliest commercial programs for the 8080 used his operating system, and from then on there really were very few attempts by others to design different ones.

Kildall's program is called Control Program for Microcomputers (CP/M). Newer versions of CP/M have been written in 8086 machine language to run on the IBM PC and in 68000 machine language to run on the most advanced microcomputers being designed 10 years after the program was first written. CP/M was also the model used when Microsoft released the first version of MS-DOS (PC-DOS), which competes with CP/M-86 for use on the IBM PC.

When CP/M wakes up in the Apple, you can't talk to it in machine language as with the Apple Monitor, but neither will it accept English-like commands in BASIC. You must respond to the "A>" prompt by typing the name of a program to be run. Some of these programs are actually part of CP/M itself which load during the boot process. The rest are either "utility" programs

which extend and elaborate upon the fundamental functions of CP/M or they are "applications" programs such as word processors or database managers which take control of the computer once they are loaded and running.

One important way in which CP/M based word processing and database management excel dramatically over the use of Apple DOS based programs is in their potential for using very large amounts of external disk storage. Those few programmers who have gone the extra mile to write really excellent 6502 DOS programs have almost universally designed them to operate strictly on Apple's own 128K disk drives. Some newer ProDOS based programs for the 6502 don't have this limitation, but you can always be sure that if you are using a CP/M database management program, you can conveniently attach floppy disks with up to 3000K bytes on each diskette or a hard disk with 5 or 10,000K bytes.

Summary

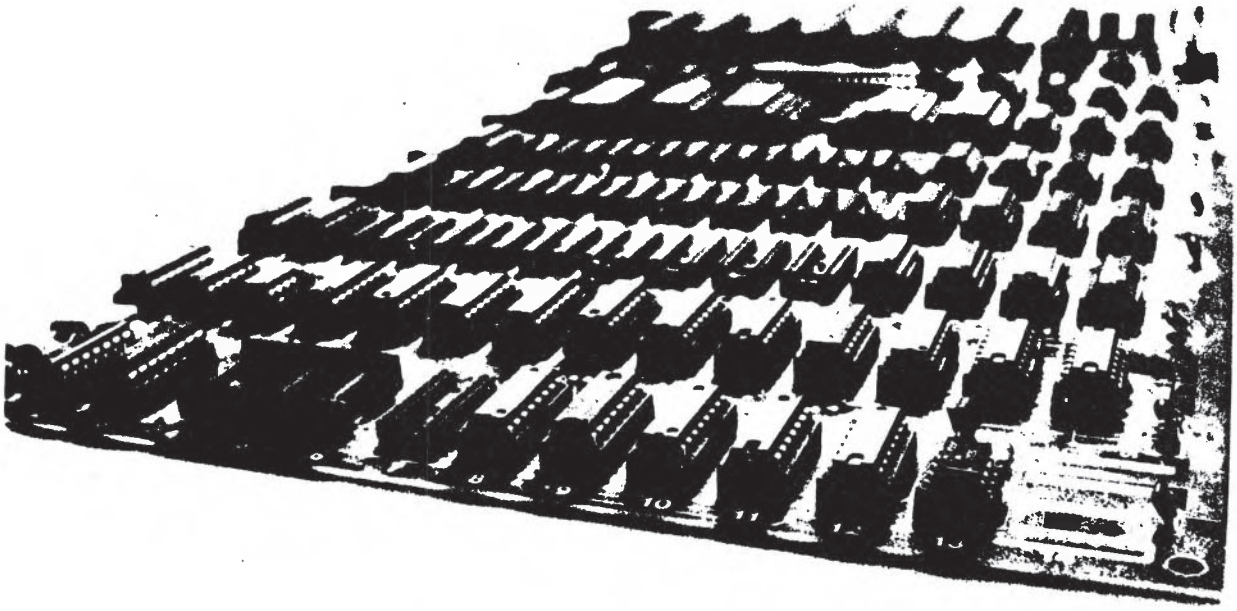
In reading the first two chapters you should have made an important transition. Well crafted applications programs and the friendly affect of Applesoft BASIC try to lull you into thinking of the Apple as a screen, keyboard, and slot into which you push diskettes. That is a perfectly fine way for a beginner to approach the Apple; it eases you into a positive relaxing relationship with the device. However, once you've become comfortable with using a computer this sort of black (and beige) box approach becomes very unsatisfying. The inner workings we've just covered are not intuitively obvious nor extremely straightforward. Nonetheless, you should be developing a mental framework which lets you understand the Apple as it really is.

With your Apple closed up, plugged in and turned on to the square bracket "BASIC" prompt, type: `CALL -151`. When you hit the Return key, you should see an asterisk and a flashing box on the screen. Now type `FF3A.FFD8` and then hit Return. The numbers and letters on your screen are the instructions by which your 6502 is able to make the sound "beep." Type `FF3AG` and then hit Return to hear what this machine language program sounds like.

PART 2

The Advanced Apple

- **CHAPTER 3 Hardware Overview**
- **CHAPTER 4 Summary of Program Options**



Chapter 3

Hardware Overview

Up until now, you have been led to believe that the 6502 microprocessor chip controls the Apple. You know that it operates at a clock speed of one megahertz and that it is responsible for moving data from place to place in the RAM memory and for moving data in and out of the Apple through its I/O ports. You've probably also heard that it can turn over control to other processor cards running at two, four, six, or even 12 megahertz. All this will be reviewed in detail in subsequent chapters.

The Video Display Generator

What you probably didn't know or haven't given much thought to is that most of the data manipulation and a great deal of the mathematical calculation that goes on in a standard unenhanced Apple takes place at a speed of over 14 megahertz. The 6502 is not alone on the motherboard.

The 6502's faster partner is the Video Display Generator (VDG) whose job is to control the electron gun sweeping along in your video monitor or TV. A great deal of the Apple's hardware design, and almost all of the trickier parts of programming an Apple in BASIC, are dictated by the peculiar needs and special wants of the VDG. In fact, during 50 percent of the time your Apple is turned on, the VDG has complete and uncontested control of the entire machine. You don't notice this time because it is simultaneous with the 6502's time. In each millionth of a second, the 6502 gets the first half and the VDG gets the second half. The 6502 is not allowed to make even the tiniest peep during the VDG's time, and the VDG gets its time even if you don't plug in a video monitor to see what it's doing.

The VDG is hard to appreciate as a distinct entity on the motherboard of the Apple II because it is made up of about 20 different chips scattered all around. In the Apple //e and //c , it's a little less cryptic, because although it is still sprawled out into five or six chips, it is embodied largely in its very own 40-pin DIP, the Input/Output Unit (IOU) which sits near the 6502 on the motherboard.

Don't get the idea that this exagerates the importance of this thing. Wozniak's patent for the Apple (U.S. Patent No. 4,130,862) is called "Microcomputer for use with Video Display." Chapters 5, 6 and 7 are devoted to exposing the inner workings of this beast, outlining its modification in the //e and //c, explaining why it does what it does, and giving you the information you need to tame it, enhance it, or override and replace it with special peripheral cards.

Input Scanners

The Apple does not sit around waiting passively for input. Rather, it constantly scans its horizon. This goes on all the time, no matter what the 6502 or the VDG are up to. The horizon being scanned is the keyboard and the device doing the scanning is a keyboard decoding chip called the AY3600 (early model Apple IIs have an MM5740).

The keyboard chip has its own clock which emits pulses at about 90 kilohertz (90,000 cycles per second). It also has a check list of 90 possible key locations, and every time it receives a clock pulse it looks at the next key location on the list. As a result, it manages to check all 90 keys about 1000 times each second. This is pretty slow compared to the 6502 and horrendously slow compared to the VDG, but it's an awful lot faster than your average touch typist and that's all that counts here.

The keyboard chip is built into the Apple keyboard in the II and the II+ and it sends fully encoded ASCII data bytes down to the motherboard via the keyboard connector cable. In the //e and //c, the keyboard chip has been moved down onto the motherboard, so the keyboard cable carries only the raw key location pulses. Further, the bytes emerging from the AY3600 PRO in the //e and //c are intercepted by an extra ROM chip and reinterpreted before being passed on to the rest of the system.

This extra ROM provides a great deal of increased versatility to the //e and //c, because several different keyboard versions can be switched in or out by turning on different parts of the ROM. For instance, when the Caps Lock key is pushed, the ROM makes the //e keyboard look like the II+ keyboard from the point of view of the 6502. Further, the use of this extra ROM makes it convenient for Apple to release special versions of the //e for "Dvorak" style keyboards and for a variety of European keyboard layouts by simply putting in a different ROM to do this last retranslation step and then painting new symbols on top of the keys.

The "keyboard" switch on the //c connects directly to the ROM chip, so pushing it instantly changes the ROM's interpretation of incoming key codes, thus activating Dvorak mode. In addition, it connects to part of the video system, so special European characters can be activated on the screen and on the keyboard simultaneously.

The Apple has a second scanning system which it uses to keep track of joysticks and game paddles. Like the keyboard, this system has a scanning frequency of about 90 kilohertz, but unlike the keyboard, there is no special independent clock. The scanning is all done by a particular set of machine language commands and requires the full attention of the 6502 for its timing.

There is an even more sophisticated hardware and software scanning system for the mouse. In the //c, that system is built-in and it uses a complex array of interrupts, softswitches, timers and flags to let the 6502 follow the mouse at the same time it does everything else it's supposed to be doing. The Apple mouse interface for the II, II+ and the //e actually uses a 6805 microprocessor to do most of the work. This system is faster and more accurate, but more expensive, than the //c's built-in approach.

You are by no means limited to using the built-in keyboard, game controls and mouse to get commands and data into the Apple. Chapters 8, 9, 10 and 11 cover a variety of alternatives, from detachable keyboards you can hold in your lap to special tablets you can draw on to microphones you can speak to from across the room.

Real World Interfaces

There are all kinds of information producing and information absorbing things out there in the world which cannot and will not understand ASCII directly. Computers (and computer owners) aren't very tolerant of being ignored by or cut off from anything, so a boom industry has developed around the process of "transducing" all kinds of information into signals a computer can understand, and, obversely, transducing computer style signals into a myriad of other kinds of output.

One important category of information is embodied in the frequency of continuously oscillating signals. The most obvious example is sound, and an Apple can be outfitted to generate speech or music, as well as to accept and interpret voice commands. The fundamental process is to use a microphone to capture a mechanical sound wave and to represent it as an electrically varying voltage. This was sufficient in the old days of radio and audio electronics, but for a computer to interpret the information, one more step is required.

For example, let's choose a wave which takes a full second to go from its minimum to its maximum voltage, and then starts heading back down toward its minimum. The computer's approach to monitoring this signal is to go out a hundred times a second and measure the actual current value of the voltage. If you recorded the measurements, you could come back later and plot out a fairly good representation of the actual wave.

With graph paper, you'd mark out the positions on the x-axis for a hundred of the one hundredth of a second sampling intervals, and you'd plot the value at each interval along the y-axis. If you connect the dots the regenerated wave appears a bit jagged when looked at from close up, but as you move back from your drawing it seems to smooth out until your eye can't pick up the difference. In fact, the quality of one of these devices is determined by how rapidly it does its sampling and how many different little incremental levels it can recognize.

The generic term for continuously varying information is "analog" and the generic term for discrete binary computer style information is "digital," so the process described in the previous paragraph has been termed Analog to Digital conversion and is usually abbreviated as A/D conversion. It is also common to say that the information has been "digitized." You'll no doubt be surprised to learn that the reverse process of using the digital data to regenerate a fair approximation of the original analog signal is called Digital to Analog (D/A) conversion.

The simplest use of A/D and D/A conversion would be as a sort of computerized tape recorder. Analog information is captured, digitized, stored, and then reconverted into analog form to reproduce the sound when called for. This is the basis of the new laser disk digital audio systems. But a computer can do more.

The human eye has the capability of converting particular electromagnetic frequencies into nerve signals the brain can interpret as color, and the human ear does a conversion of sound frequencies into nerve signals the brain can interpret as tones and pitches. Similarly, a skilled programmer can design analysis software that gives meaning to the digital patterns captured by an A/D converter. It is possible to capture a spoken word as its digital representation and then subject the captured data to an elaborate mathematical analysis so that the pattern can be recognized, assigned meaning, and an ASCII representation of the spoken word passed on to the 6502.

A/D and D/A conversion is important for a large variety of scientific laboratory tasks, for operation of robots, for capturing images from a video camera, and for controlling any number of day-to-day devices such as light switches and air conditioners. The A/D and D/A devices available for the Apple range enormously in quality and sensitivity. These will be reviewed in some detail in Chapters 14 and 15, along with some fundamentally digital devices which monitor time and count discrete events. The Apple has its own set of four built-in A/D converters which are most commonly used with game paddles or joysticks, but they're not particularly fast or sensitive.

Data Transfer Between Digital Devices

The simplest way to move programs or data from one computer to another is to write the information onto a disk, pull the disk out of the drive and put it into the second computer. This simple approach fails when the second computer can't read Apple disks, the second computer is far away and you can't wait for the disk to be sent in the mail or carried across town, or you want to communicate with a large number of nearby computers on an instantaneous basis.

The first problem, disk incompatibility, is fairly serious. Apple disks are unique and no other type of computer can read a disk written by a standard Apple drive. An Apple can be hooked up to a large variety of non-Apple disk drive and magnetic tape systems to permit interchangeability, but this approach is often very expensive and usually requires a special setup for each different machine you need to communicate with. The second and third problems come down to questions of time, money and efficiency.

In all of these cases, the problem is solved by running wires between the devices and letting them communicate directly. This, however, introduces an additional range of compatibility problems having to do with exactly what kinds of electrical signals will be used, when information will go in one direction or another, etc.

The Apple //c dodges most of these problems since it has everything that you need already built-in. However, an Apple II, II+ or //e as shipped from the factory has essentially no ability whatsoever to be wired up to a second device. Let's start with a really fundamental second device, a printer (OK, OK, a printer isn't actually a computer, but it looks like one from the point of view of accepting data and these things almost universally never have their own disk drives). There's no way to connect a printer to a "bare bones" Apple.

The good news is that Apple and dozens of other manufacturers have designed and built peripheral cards you can plug into the Apple's expansion slots which provide the ability to communicate. The bad news is that there are dozens of different cards on the market, most of which differ in a variety of features. To pick the proper card to buy, you have to know either the exact needs of the two devices you need to interconnect or you have to pay extra money for a single fancy card which can handle a broad variety of very different tasks. Chapters 16, 17, 18 and 19 are devoted to explaining just exactly what these interfaces are supposed to do, and how to go about picking the correct ones for the task at hand.

Communicating with a printer is usually so straightforward that once you purchase the correct card for your printer, you're ready to go without further fuss. Almost any other sort of communication requires both hardware and additional purchased software to manage the two-way information transfers.

The two broadest categories of communication protocols are serial and parallel. As a review, serial communication is conducted along a single wire with the eight bits in a byte following along one after another, while parallel communication uses eight wires side-by-side so the eight bits in a byte can travel together.

Parallel connections are usually used inside a computer system and to link two devices which are fairly near each other. Serial connections are used for communicating over the phone lines and for communicating between devices which are more than a few feet apart within a single building.

The most elaborate type of parallel communications involves a system called the IEEE-488 bus. This is a means of making a number of devices feel as if they were actually sitting inside a single computer. Individual devices on the IEEE-488 bus must be no more than 12 feet apart, and a string of 15 devices must be squeezed into a total cable length of no more than 60 feet. It is a versatile but rather expensive system and its use is largely limited to specialized laboratory systems.

The major use of parallel communications for microcomputers these days is for connections to printers. In selecting a parallel interface card for a printer, you still have to do a good bit of checking to make sure the printer and your card agree on which wire in the cable will carry which signal, although one particular configuration called the Centronics standard is becoming dominant.

All of the common types of parallel connection use a standard five volt signal which is directly comprehensible to most of the chips in a computer, but because this is a fairly low voltage, it is not powerful enough to be used in long cables. In serial communications it is customary to use 12 volt signals which can carry over a longer distance, or to use a different electrical property, "current flow", for very long cables.

Many older communications devices, most notably the venerable teletype, use these current-based signals, and a whole legion of laboratory and business equipment which used to get connected to teletypes still provide their communications signals in this form. The most common arrangement is called "20 milliAmp current loop" and Apple makes a card which can accept and send these signals. Early in the days of Apple it was important for Apple to be able to sell the Apple II as a replacement for the old time teletype, but this communication mode is rapidly disappearing in the market place.

The current dominant star in communications is a kind of signal and a set of protocols which are together referred to as RS-232C. This is a system which uses serial signals at 12 volts, and in which a number of control and signal lines are lined up next to the data lines. RS-232C serial connections are used for printers and modems as well as for directly connecting terminals to mainframe computers. The disadvantages of RS-232C systems include the limit on the length of the cable to about 50 feet as well as the limit on communication speed to about 19,200 bits per second for direct machine to machine connection (for a variety of reasons, RS-232C is rarely operated faster than about 1,200 or 4,800 bits per second). RS-232C is popular because these speeds are near the limit of the ability of printers, telephone lines, and CRT display screens for accepting data and because telephone lines can be used for occasional transmission over longer distances.

The Apple //c has two complete RS-232C ports and no provision for parallel communications. This choice was due in large part to the need to avoid the radio frequency interference generated by the parallel communications cables and in part by the greater versatility of serial RS-232C communications systems.

One new rising star in communications is a serial connection system called "RS-422." It requires just two wires, supports cable lengths up to 2,000 feet, and can transmit data at nearly one million bits per second. This standard is important because it makes it possible to establish Local Area Networks (LANs). Dozens of computers in a building can be connected to each other and can exchange messages nearly instantaneously without paying a penny to AT&T (formerly Ma Bell). There are actually several competing hardware strategies to pick from, some of which can carry on communication at rates of 10 million bits per second; these are explored in Chapter 20.

Apple Memory and the $C000 Space

The true path to understanding the Apple for most programming and hardware tasks leads through a range of about 4K of memory addresses which we can call the "$C000 Space" (pronounced see-thousand). This is a range of addressable locations the 6502 can use for a variety of tasks related to input and output. Some are used to operate switches, some are physical "ports" to the exterior, and some are used for special programs in ROM to operate peripheral cards.

Since we're going to be talking about small chunks of memory here, it will be convenient to start using a new term, the "page" of memory. A page is 1/4K of memory addresses (256 locations to be exact). The idea of a page of memory is very handy for programmers who use the hexadecimal number system since the hexadecimal representation for 1/4K or 256 is $100. In this notation, the first page of memory has locations between $00 and $FF, the second page has $100 to $1FF, the third page from $200 to $2FF, etc. This breakdown of memory into pages is actually meaningful to the 6502 at the level of its machine language instructions.

The most interesting and subtle set of locations all occur in the first half of the first page of the $C000 Space, an area we can call the "$C0 Page" (see-zero). This set of 128 locations is considered a little bit tricky by even the best Apple programmers, but you may need to use them in even the simplest BASIC program. These are the objects of most of the "PEEKs and POKEs" you've heard about. There are a number of good published descriptions of the $C0 Page and the one that appears here in Chapters 22, 25 and 26 should also help. There's a lot more to this region in the //e than in the II/II+, and in the //c it has yet again more additional fine points, so you have to take care where you get your information. The functions performed here include operating and monitoring all of the Apple's built-in I/O devices.

Control Switches and Ports

When you're working in the lower half of the $C0 Page, you have to be aware that most of the locations have no ROM or RAM. Most of the locations are actually used as switches and "latches." A latch is a device which can hold a byte of data and then release it to the databus when the 6502 hits a switch. This is a popular means of passing information back and forth between two computing devices which are not completely synchronized.

Some of the locations are actually involved in communication between the 6502 and other subsystems right on the motherboard. These are often addresses which the 6502 can write to but which it cannot read. They make sense when you appreciate the presence of the 6502's co-resident, the VDG. Although the 6502 can't read from them, the VDG can respond to some of them. The 6502 can communicate with the VDG by altering one of these locations and then relying on the VDG to look in on those locations from time to time. These "softswitches" and "toggles" are actually output ports to this "other machine" sitting on the same motherboard.

In addition, the //e and //c have a few of these locations which can't be read by the 6502 or by the VDG, but which are used by the third of the //e and //c's great 40 pin DIP chips, the Memory Management Unit (MMU). There are a few more of these odd creatures which will get covered later on including one that only a cassette tape recorder can hear (II, II+ and //e and one that you can hear directly because it makes the Apple's speaker go "click."

In the II and the II+, the upper half of the $C0 Page is devoted to use as a series of control switches and ports for peripheral cards plugged into the expansion slots. The same concepts of control and monitoring still apply, but, in the II/II+, all of these 128 locations were used to operate I/O devices installed in the slots rather than built in at the factory. However, as Apple progressively adds more built-in features to newer models, little chunks of this half page are rendered into locations for built-in devices. So the distinction between the lower and upper half of the $C0 Page is not hard and fast.

ROM Areas in the $C000 Space

The remaining 15 pages of addresses in the $C000 Space are devoted to reading special ROMs that manage I/O. Most manufacturers of peripheral cards for the Apple write 6502 machine language programs which operate their cards. The tradition is to store this program in a ROM chip, build it into the card, and make it available for easy use by a programmer. You can usually cause one of these programs to be executed by typing the appropriate PR#n statement.

There is, however, something strange about the memory addresses for these ROM chips. Although this part of the $C000 Space has just 15 pages of addresses, it's possible to plug in peripheral cards carrying a total of about 71 pages worth of ROM chips (seven cards with nine pages per card, and one more card with eight pages). An Apple II with no peripheral cards has no ROM at all in the $C000 Space. An Apple //e has a full 15 pages of ROM in the $C000 Space already there when it's shipped from the factory, but it still doesn't seem to mind very much if you plug in an additional 71 pages. You end up with 86 pages of ROM locations sharing just 15 pages of addresses.

This is a "space/time" phenomenon. A range of memory addresses can be thought of as a space. The trick for having many different ROMs share their addresses is to let them be in the same space but to require that they be there at different times. Although the ROM chips do not physically move, they do move "logically" in and out of the $C000 Space so that no two of them are in exactly the same place at the same time.

The 6502 keeps looking at the same logical space. One moment it sees one set of ROM chips, but when it looks again to the same memory space a few instants later it sees a different set of ROMs. The various ROMs are switched in and out of the $C000 Space under control of programs. However, you've probably done this switching yourself dozens of times without even realizing it. When you type PR#3 to turn on a video card, you are switching the video card's control ROMs into the $C000 Space.

The ROM area of the $C000 Space in the //c is much simpler. The machine comes with 15 pages of ROM and that's the way it stays. There is one interesting twist though. Most of the machine language programs in those //c ROMs are written in extended 65C02 machine language rather than in 6502 code.

Iron, Plastic and Permanence

During the past 10 years, there has been a steady and progressive effort to store data as smaller and smaller patches of magnetic marks on plastic and to put more and more concentric rings of these marks onto the same flat disk. Five and a quarter inch floppies which could store 80K bytes have given way to 5 1/4 inch floppies which can store 2,400K bytes and to 5 1/4 inch hard disks which can store 500,000K bytes (yes, that's 500 megabytes), all of which can be used with an Apple. Chapter 23 is a detailed exploration of the design of these devices and of the engineering challenges involved in expanding their capacity.

The recent development of small inexpensive lasers may ultimately lead to the replacement of the electromagnetic system. The new optical disk technology depends on using electrical pulses to generate tiny, focused, and intense beams of light which can burn small holes in a plastic recording medium. Lasers are used to read by shining a weak light on the disk and watching for the reflection to fail when the reading beam passes over a burned spot. Laser recording promises much higher densities than can be achieved with magnetic technology and also an escape from the tendency of magnetic media to be not quite as permanent as you would sometimes like. It is still very difficult, however, to erase and rewrite disks using laser technology.

In Chapter 24 we'll try to swing through a scan of what's out there on the market in semi-permanent storage devices. Choice of the right device for you depends on a whole variety of factors such as interchangeability with other computers, reliability, and the availability of purchased software which can work with a particular kind of disk device.

MICROSOFT
PREMIUM SOFTCARD IIe
MICROSOFT
PREMIUM SOFTCARD IIe
Programmer's Manual
OPERATING SYSTEM USER'S GUIDE
DIGITAL RESEARCH
CP/M Plus
Operating System
DIGITAL RESEARCH
Programmer's Utilities
INTRODUCTION TO WORDSTAR
MicroPro

Chapter 4

Summary of Program Options

Assembly Language Programming

Most programming ultimately consists of stringing together various chunks and pieces of the operating system software and arranging for your own data handling needs. The easiest way to do this is by programming in a high level language such as BASIC or Pascal; for the most part, the language interpreter or compiler knows when to call the operating system services and what to tell them to do. If, however, your program must run extremely fast and if you've got plenty of time to do the programming, you may want to handle all the details yourself, in person, by programming in assembly language.

To do assembly language programming you have to have a fairly good grasp of the detailed organization of your computer's hardware, because most of the instructions refer to actual hardware locations. This knowledge must include the sort of details about the 6502 which are reviewed in Chapters 27 and 28, and also a comfortable feel for the various landmarks scattered around the Apple's address space.

The language itself is usually fairly simple because most microprocessors have relatively small sets of instructions. The process of learning involves finding out how to put together clumps of instructions which cause the events you want to happen. There are many books which describe the instruction sets of various microprocessors, but the 6502 and the Apple II are also the subjects of a very large number of books about how to put these instructions together into productive programs.

In Chapters 29 and 30 you'll see that the 6502 instruction set is fairly sparse compared to some other microprocessors. However, even though it is sparse, and even though it is an older microprocessor, it is absolutely the microprocessor of choice for an assembly language novice because of the large number of well written books for beginners and the variety of assembler programs and utility software available to help you learn to program.

The //c comes with a new 65C02 microprocessor which has a larger and more powerful instruction set than the original 6502. The expanded instruction set includes all of the 6502 instructions plus several new ones. This means that you can take advantage of all the older books and guides, but then you can go on to write faster and more compact programs using all the new 65C02 instructions.

This book does not attempt to teach you how to program in assembly language and it does not provide full explanations of all the instructions. However, it does provide an overview of

the process of assembly language programming. Also, Chapter 31 reviews several full featured assemblers you can use to convert your assembly language programs into machine language.

If you decide to learn assembly language programming you will immediately discover that you're expected to own an assembler before you even start. The assembler plays the role of the Applesoft Interpreter; it accepts your assembly language instructions and generates machine language. Your choice of an assembler will have a very substantial effect on how easy it is for you to learn assembly language. Hopefully the information in Chapter 31 will get you out of the chicken and egg problem of choosing an assembler before you really understand what an assembler does.

Managing Programs, Disk Files and I/O

The manufacturer of any computer usually supplies a collection of machine language programs which are together called an operating system. The programs in an operating system are responsible for managing the I/O tasks such as reading data on the disk drives, for causing programs to be executed by the microprocessor, and for keeping track of where programs and data files are stored so they can be conveniently retrieved from storage when needed.

There are currently five different, well-supported operating systems available for the Apple II, and each of them comes in several different flavors. The first and foremost of these comes in two parts, a machine language level "Monitor" which is stored in a ROM in every Apple, together with the well known DOS 3.3 distributed on disk. The second major resident operating system is the Pascal operating system, and it essentially starts from scratch making little use of the routines provided in the Monitor ROM, and completely replacing DOS. The third is CP/M for which several companies have written versions of the I/O routines together called the BIOS, and the fourth is MS-DOS 2.0 which runs in its own independent hardware environment in the Rana 8086/2 peripheral. Fifth and newest is ProDOS. All of these store their information on disk in completely unique fashions, so you cannot even use a blank disk which has been formatted by one of the other systems. Chapters 35, 36 and 37 step through a detailed comparison of several features of DOS 3.3, CP/M, ProDOS, and MS-DOS 2.0.

DOS and the Monitor are sort of indigenous to the Apple; they grew up piece by piece from Wozniak's fertile mind as the earliest Apples took shape and they reflect his ideas of what an operating system should do rather than any standard industry model. The greatest asset of this operating system is that it is easy for the beginning programmer to use. All of the commands are given in simple statements which have the feel of programming in BASIC. Further, these commands provide a truly incredible ease of control over all sorts of varied I/O devices. This ability is absolutely unparalleled in any other microcomputer operating system.

The weaknesses of the DOS/Monitor operating system come in three categories. The first two stem from the very fact that this is essentially a BASIC programmer's operating system. Machine language programmers who are trying to design top quality, high speed commercial software are substantially left in the lurch. Using the subroutines available in the operating system without actually giving commands in BASIC is not at all straightforward. Worse, Apple has never made a substantial attempt to publish a clear set of procedures for using the operating system features from machine language programs.

This has spawned a bizarre dichotomy among skilled Apple programmers. One bunch has tended to write a minimal skeleton of every program in BASIC, to loop out of BASIC into machine language for fast subroutines, then to come back in to issue operating system commands. The second approach has been the development of a feverish "hackers" network of pure machine language programmers who go about trying to "crack" the operating system and who occasionally publish personalized tricks for making it behave in one of several computer magazines.

Unfortunately, becoming a master at Apple machine language programming doesn't help much when you try to program a different computer. Further, since Apple doesn't intend for the operating system to be used by beginning machine language programmers, it has changed the system a number of times in such a way as to force machine language programmers to do substantial rewrites. The best Apple programs tend to be so specialized that little attempt is ever made to make them run on any other computer system, and some can only run on a single version of the Apple II. VisiCalc is one of the few examples of an Apple DOS based program which was completely rewritten from scratch to run on the IBM PC.

The second weakness of the DOS/Monitor system is that it is a "command driven" system. A computer novice must learn the commands and understand their effects before he or she can use any of the features, such as running a program. This is why it has been popular for Apple software to be sold in an automatic "boot and run" format which never requires the user to interact with the operating system.

The third weakness stems from its uniqueness at the level of floppy disk usage. The renowned zero interchangeability of Apple text files with text files on diskettes from other computers is due in substantial part to this. In fact, some of the things DOS does are so unique that it may not be possible to "boot" a disk unless you are actually using a genuine Apple Disk II disk drive. You cannot even count on substituting a non-Apple disk drive made by the same equipment manufacturer.

In Chapters 33 and 34 there is a fairly detailed overview of what DOS and the Monitor do, how they do it, and where you can learn more. In spite of the weaknesses mentioned above, this has been the dominant Apple operating system and one you're likely to become familiar with. It is the one given out free from 1978 to 1984, and the one that runs the huge assortment of excellent Apple software.

Pascal Operating System

The chief attractions of the Pascal Operating System are that it constantly provides menus that describe your options before you give a command (see Figure 4.1), and that it is extremely gentle with the novice, constantly double checking and warning you in a calm manner when it thinks you've ordered it to do something you didn't mean to. Sounds wonderful, doesn't it? Well, all this has its price.

First, it tends to be a bit sluggish when compared to DOS, and second, you will very soon wish that it would stop being so friendly and protective all the time and simply let you go directly about your business. Third, and most serious, this is an operating system for people who want to program in Pascal or Modula-2. It is not for people who want to do word processing or play computer games, etc. For a variety of reasons, there has never been very much commercial software written to run on the Pascal Operating System. The few commercial Pascal programs on the market such as PFS and Statpro hide the fact that they're using the Pascal

Operating System so you never get to enjoy it. Further, this is not the operating system for people who wish to learn to program in machine language. The routines which make up the operating system are not at all easy for a machine language programmer to use.

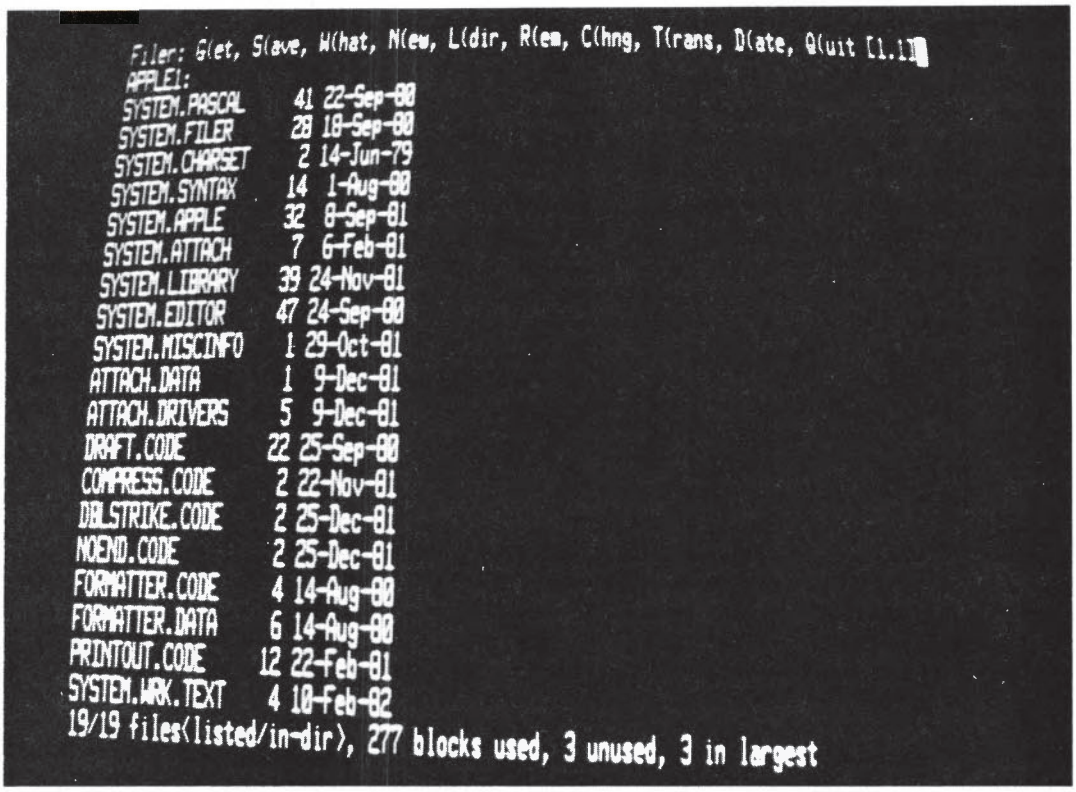


Fig. 4.1 Pascal directory listing. The Pascal operating system can date stamp its files, and it always prints a menu of options across the top of the screen.

The future may see some shift towards increased importance of Pascal programming on the Apple because of the growing emphasis on portability of programs among different microcomputers. Apple Pascal 1.1 is also known as UCSD Pascal II.1. It is an extension of the original UCSD Pascal. Softech microsystems has achieved essentially complete portability of programs among all microcomputers with its own different version of Pascal called the UCSD Pascal IV.1 p-system, and this version is now available to run on the Apple. It is important to note, however, that this portability is only meaningful at the level of programming in the high level Pascal language. There is no particular disk compatibility among computers using IV.1, and another consideration is that it executes more slowly than Apple Pascal 1.1.

CP/M 80

To use CP/M you have to purchase some sort of peripheral card or the //c expansion module which has a Z-80 microprocessor system, but the rewards are substantial for the serious microcomputer user. A novice will find CP/M to be the most confusing of all the operating

systems. Further, it is very weak at handling errors of any sort—mistakes or misstrokes often result in an unresponsive or "hung" system, and CP/M is remarkably inarticulate in telling you what you did that it objected to. The version of CP/M distributed with the Appli-Card from PCPI is a little better in this regard than the one you get with the SoftCard from Microsoft.

However, once you get past your first few lumps and bruises, CP/M will impress the DOS or Pascal user by its speed and no nonsense efficiency. The real reward, though, is in the top flight software available for word processing, database management, and other business applications.

One important reason for the strength of the software is that CP/M is a dream operating system for machine language programmers. It is very well documented and it is arranged specifically for the convenient use of skilled machine language programmers. It comes in several parts, and the real key to its success is that Digital Research sells a sort of programmer's environment which is the same for an enormous number of microcomputers using 8080, Z-80, 8088, or even 68000 microprocessors (almost anything except the 6502). This environment includes a list of some 50 or 60 different kinds of I/O or file management tasks a programmer might like to do and it assigns each task a number and a protocol for accepting special instructions. The manufacturer of the computer system is responsible for writing the actual machine language routines for its own system and linking these programs into the numbered tasks or "services."

The result of this is that a programmer can reasonably plan to write a fast, efficient machine language program which will run on any one of a family of computers including all 8080, Z-80, and appropriately equipped Apple computers. If he or she wants the program to run on an IBM PC or a 68000 based computer, the machine language will have to be rewritten from scratch, but since a majority of the work is actually done by CP/M services, the general structure of the program can remain the same.

MS-DOS and CP/M86

There are several peripheral boards available for the Apple which use the 8088 to run MS-DOS 1.1 and CP/M86. These two operating systems are essentially identical to CP/M except that MS-DOS 1.1 lacks much of the variety of commands available with CP/M. Further, at this writing, there is very little software for MS-DOS 1.1 which is not available in identical form for Z-80 based system, and the new Z-80B cards often run the programs faster than the 8088 can.

The most serious current reason for not choosing these operating systems is that most of the software that runs on the IBM PC will only run on the PC itself or on carefully crafted PC-compatible machines. The various 8088 cards which are currently available for the Apple were all designed before the PC software market had fully shaped up, and so they made no particular effort at providing PC compatibility.

However, the story is a bit different for MS-DOS 2.0. There are several programs for this newer operating system which are designed to use up to 256K of RAM, and these can be very interesting, especially for the business user. The 8086/2 from Rana is a complete PC-compatible machine in a box, including PC-compatible drives, PC-compatible video, and 256K of RAM. It runs MS-DOS 2.0 and has the support of both Apple Computer and several major MS-DOS software publishers. It is actually a very powerful machine which exceeds the power of most PC-compatibles on the market as well as presenting graphics and processing features not available for the PC itself.

ProDOS

This is a totally new operating system for the Apple II family of computers. It is easy for non-programmers to use because most of its features are available from menus. At the same time, time, though, it has a very well designed "machine language interface" with well documented system calls in the style of CP/M. Even the Applesoft interface has some powerful new commands. Thus it has something new and improved to offer for a wide range of potential users. In addition, it will work comfortably with nearly any kind of new external mass storage device, thus breaking free of the 128K Disk II.

ProDOS's closest relatives are the Sophisticated Operating System (SOS) of the Apple III and the UNIX operating system originally developed at Bell Labs. The similarity to SOS results in substantial disk compatibility between the Apple II and Apple III, and the similarities to UNIX result in convenient management of hard disks and interesting new ways of carrying out I/O tasks.

Programming Languages and BASIC

If you want to learn to program, you're going to have a pick a language, then go read some books about your language, do a little programming to get the feel of it, and then start the long gradual process of becoming eloquent. BASIC (Beginners All-Purpose Instruction Code) and its older cousin FORTRAN (Formula Translator) dominate computer programming. FORTRAN was born in the dawning days of the computer age (1956) and is so well known and has been used in so many programs that it still has an enormous influence. Until the last year or two, it has remained the unchallenged language of choice for scientific programming because it is possible for a scientist to make use of huge libraries of FORTRAN subroutines written by nearly two generations of programmers. FORTRAN compilers are still among the best selling language packages.

BASIC closely parallels FORTRAN's syntax and organization, but is a bit less stiff and formal. This language is also getting a bit grey around the ears since it has just celebrated its twentieth birthday. However, there is no great library of BASIC subroutines because BASIC comes in so many flavors and because its freewheeling structure and popularity in "interpreted" form aren't well suited to plugging in other people's subroutines.

For an Apple user, there are typically five choices among the various BASICs. The Apple is sold with two versions, Integer BASIC and Applesoft BASIC (see Figure 4.2) and CP/M users can choose among MBASIC, GBASIC, and CBASIC.

There are three principal areas where Applesoft excels over Integer BASIC. The first of these is implied by the name "Integer." This language can manipulate the 65,535 numbers that come between -32,767 and +32,767 and that's it. Applesoft is also called "floating point" BASIC because it accepts numbers with decimal points and can use scientific notation to handle an infinity of real numbers over a range of 10 to the 64th power (which is plenty).

The second difference is that Applesoft has greatly enhanced capabilities for playing around with strings of characters, and the third difference is Applesoft's ability to conveniently manage Apple's high resolution graphics. Integer offers a few extra services for the programmer, but all of these capabilities can be added to Applesoft by loading in some popular utility programs. Integer is important only because it was the first BASIC for the Apple and many Apple programmers cut their teeth on it. Apple still distributes Integer BASIC so that all Apple users will be able to run purchased programs written in Integer BASIC.

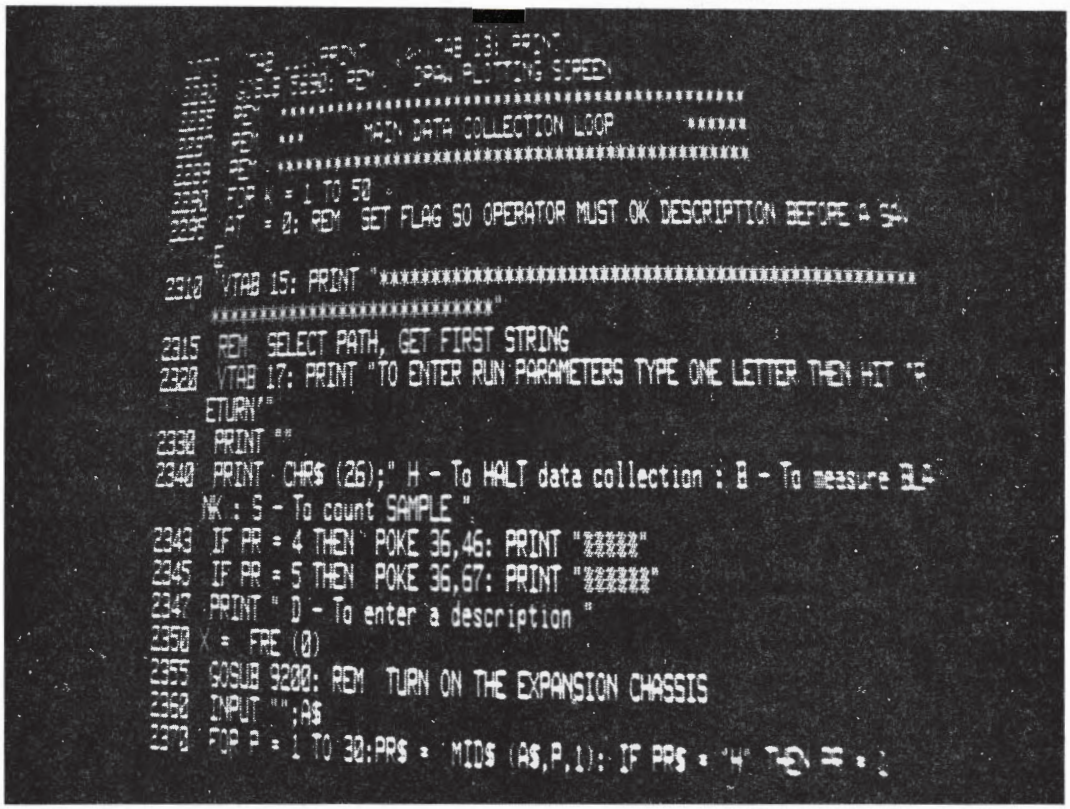


Fig. 4.2 Applesoft BASIC program listing. This is the most popular dialect of BASIC for the Apple.

The three CP/M BASICs are similar to Applesoft in many ways. CBASIC differs in that it is a compiled version of BASIC. Compiling can produce faster running programs, but this yields no special advantage since a program written with the Applesoft interpreter can be compiled after it's been completed by any one of several commercial utility programs.

GBASIC is largely identical to the standard CP/M MBASIC except that MBASIC does not have graphics commands. Since Wozniak has built a very well documented Video Display Generator into the hardware of the Apple, it was a relatively straightforward task for Microsoft to duplicate the high resolution graphics commands in Applesoft, add them to MBASIC, and call the new hybrid GBASIC. One of the principal arguments in favor of using MBASIC is that programs may be portable among CP/M and CP/M86 systems, but if you use GBASIC, portability is lost and you might as well be using Applesoft.

In the discussions of operating systems it was pointed out that the I/O routines in the DOS/Monitor system were easily accessible from Applesoft BASIC and that the I/O routines in CP/M were easily accessible from 8080 machine language. When you program in MBASIC, you discover that it is very difficult to take advantage of the Apple's formidable I/O capabilities, and worse, it's also very difficult to use CP/M's operating system services. The beginning programmer will have far greater power over the machine if he or she chooses Applesoft. A skilled programmer who is comfortable with both 6502 and 8080 machine language will have full use of the services of both operating systems—but we're talking here about a very, very select crowd.

If you choose Applesoft BASIC, you can begin with the Tutorial and the Programmer's Manual sold by Apple, and then supplement that with any one of dozens of books on Applesoft. However, it's after you've gotten into it a bit that the real fun starts. There are literally hundreds of "utility programs" and explanatory articles out there which you can use to make Applesoft jump through hoops and turn somersaults. Learning Applesoft BASIC is a continuing process. It can begin as an easy convenient language for learning to do simple tasks, but it can lead you into the heart of hearts of microcomputers.

This book does not include a section on learning to program, but Chapters 38 and 39 are intended to explain the lay of the land for advanced Applesoft programming. Many programmers who don't want to become advanced but find themselves getting into troubled waters may want to pick through the information in Chapters 40 and 41 on handling memory space problems and speed problems in Applesoft programs.

Structured Programming and Pascal

Your choice of a programming language should also take into account the kind of task you hope to be able to accomplish. There are two major areas where BASIC and FORTRAN are subject to problems; the first is in the construction of very large and complex programs and the second is in the area of writing operating system level software for managing I/O. You may not be planning to write large programs, but be forewarned that these things do tend to grow rapidly once started.

Pascal is a relatively young language (1971) which addresses the weaknesses of FORTRAN and BASIC by building large programs from a series of smaller units. Each unit is composed of a nested hierarchy of functional program segments and can be planned and written separately as a logical, coherent entity before the various parts are all linked together when the program is compiled. By making the program out of understandable little chunks, Pascal tends to make it easier to add, substitute, or modify units long after a first version of the running program is completed.

The Pascal compiling process has some serious limitations however. Pascal is not actually compiled into the machine language of the 6502. Rather, it is compiled into a machine language called "p-code" which cannot run on any real microprocessor. The Pascal operating system is written in real 6502 machine language, and it creates, from software, something known as the "P-machine." This machine does not exist as hardware. It is a mental notation. The p-code generated by the compiler is the machine language of the p-machine. The p-machine executes the p-code instructions at run time by interpreting them into 6502 machine language.

Pascal was designed in this fashion so that p-machines could be created on every kind of computer in the hope that compiled p-code would always encounter a seemingly identical p-machine no matter what computer was actually being used. There are three flies in this idealistic ointment. First, Pascal is slow because it goes through so many layers of translation. Second, the p-machines which have been created on various computers ended up being so different from each other that you cannot expect a program to actually be portable (unless all the machines use the UCSD p-system Version IV.1 from Softech). Third, this multi-level infrastructure tends to make the programmer completely isolated from the actual hardware of the microcomputer. If an I/O facility is not provided by the language directly, you will have a very difficult time trying to do the job yourself. Further, the final running machine language tends to be comparatively inefficient.

This is all acceptable in the classroom where students are learning good program design, but it has made Pascal fairly unpopular with commercial software publishers. With the introduction of new microprocessors such as the 68000, however, the computer can make up for the sluggishness and inefficiency by sheer computational brute force and speed. As a result, Pascal is much more important for commercial programs for 68000 based machines than it has been for older microcomputers.

The C programming language was written (1974) by people who liked the ideas behind Pascal but wanted to write efficient commercial and operating system software. C retains the structured, unit based approach to programming and even looks a bit like a Pascal program when it is written out. However, it is very hardware oriented. The programmer can use many commands that resemble assembly language more than anything else. The result is that when a C program is compiled, it produces extremely compact and efficient machine language. In fact, it is nearly as efficient as code written directly in assembly language. C is not a language for beginners, but its spirit is so close to Pascal's that it is fairly accessible to programmers who have learned to program by using Pascal.

Modula-2 (1982) is one of the newest computer languages, but it is causing quite a bit of excitement. It was written by Niklaus Wirth, the principal creator of Pascal. It is essentially an update of Pascal for the benefit of professional programmers. It improves on Pascal's proclivity for modular structure. Pascal tends to nest its units, one within the next, within the next, whereas Modula-2 tends to make them truly separable distinct units.

LOGO

For most Apple owners, LOGO is associated with Turtle Graphics. With these two parts working together, young children can learn to program the Apple to carry out drawings of complex geometrical shapes and just plain pictures. This is an excellent use for LOGO, but it distracts some programmers from the real power and importance of LOGO. This language differs from BASIC and Pascal in that it has its roots in the science of artificial intelligence.

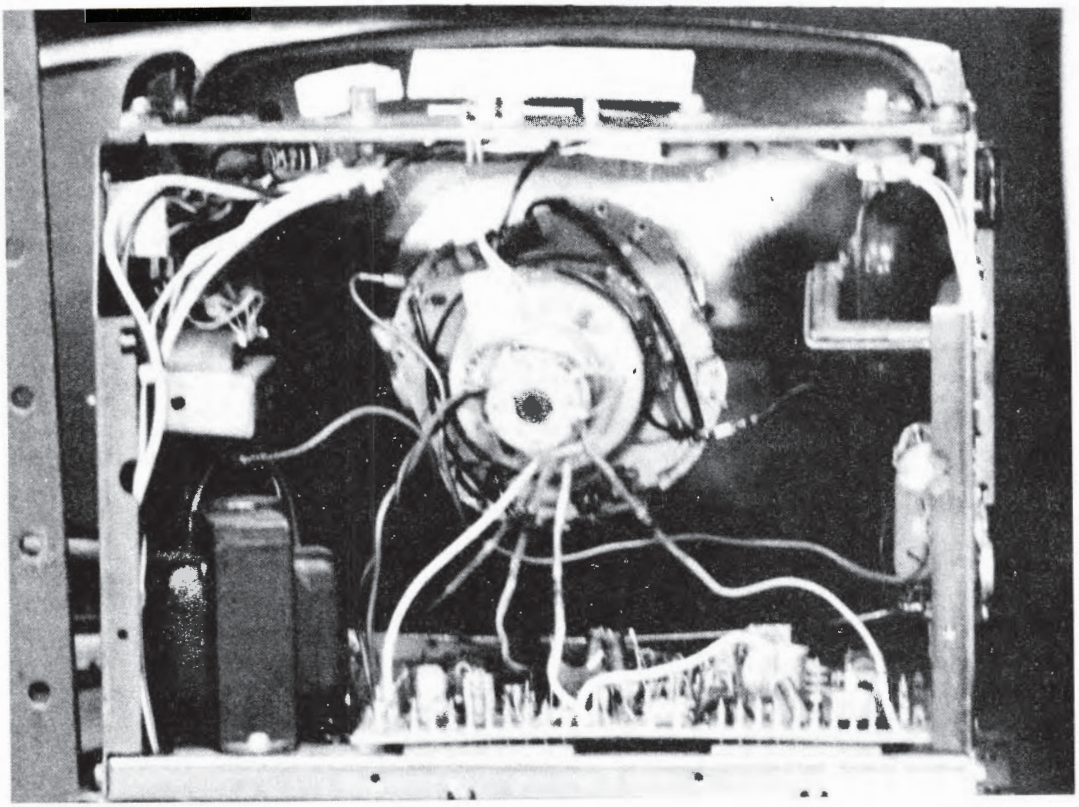
On a practical level, the most important distinction of LOGO programming is that you can use words as symbols for complex program operations. In effect, you define new commands in the language as you go along. It might seem that verbal symbols are just a crutch for the inexperienced, however, there is a good reason why a seven-year-old can write a LOGO program to construct a complex geometric shape that most adult programmers could never complete in BASIC. This mental elegance provides an interesting sort of programming power with which more programmers should become acquainted.

II

INTERFACE APPLE

PART 1

The Video Apple



Chapter 5

The Video Display Generator

On the glowing surface of your Apple's video screen, two great American passions meet: television and microcomputers. When Wozniak was designing the first Apple II in the mid 1970s, however, television reigned supreme and the passion for microcomputers didn't exist yet. Consequently, Wozniak set out to design a piece of equipment which was as much an odd new sort of television camera as it was a microcomputer.

Like any good television camera system, the Apple II produces a signal which can be sent to the antenna of any black and white or color TV set in the country to reproduce what the camera is looking at. In a standard television camera, a camera tube sweeps through a repetitive scan and captures the patterns of light and dark it sees before it. The Apple's video generator performs an almost identical kind of sweep, but instead of looking at people or scenery, it scans a region of the Apple's memory where an image pattern of ones and zeros has been placed.

Again, as with a good television camera, it is possible to change which area of memory the "camera" is looking at. In the Apple //e and //c there are six different regions or "display ranges" which can be scanned. The programmer has the role of a scene designer or director—exercising decisive control over what graphical patterns or letters and numbers get put into each of the display ranges for the camera to see.

Hardware Video Systems

It is useful to think of Apple Video in terms of three distinct systems: text, graphics and color. Within each of these there's a fair amount to be learned about both the built-in and optional enhancement hardware, and about the programming needed to control what gets displayed.

The first of the three systems is responsible for the display of text and data on the screen. This is the way in which the users and designers of word processing, spreadsheet and database programs tend to think about video. A large part of this chapter will therefore be devoted to the Apple's built-in hardware and software for managing text on the screen, and it will cover equipment devoted to improving the quality and quantity of "alphanumeric" characters you can see on the screen. It will also cover the options available for easing eye strain for people who spend many hours staring at the video representation of text and data.

A second and substantially different approach to Apple video concerns what is called "bit-mapped graphics." This is the realm in which Apple video is closest to standard television, yet the management of computer graphics is so much more subtle and complex than the management of text characters that graphics video has lagged behind text video in terms of

quality and versatility of software. Graphics capabilities are important to businessmen who need graphs and charts, to scientists who need visual summaries of mathematical expressions, and of course to artists and game designers.

Color video is a sort of third realm, although for the most part it is used to enhance graphics displays. The use of color in computer graphics requires attention to a special set of hardware enhancements, and it introduces a few more complexities into programming.

The Monochrome Video Screen

The inside surface of your video screen is coated with a material called a "phosphor" which glows briefly when it is struck by a beam of electrons. At the far end of your monitor is an "electron gun" which shoots electrons at the screen in a sharply focused beam. Left at peace, the stream of electrons would zoom straight toward you, smash to a halt at one very tiny spot in the center of your screen, and cause a single tiny bit of phosphor there at the center to glow, surrounded by an unbroken field of black.

Electron Beam Raster

However, surrounding the electron gun is a yoke made of thousands of turns of fine copper wire. In an endless ritual enacted an infinite number of times since the dawn of television, the wires of the yoke generate an ever shifting electromagnetic field. This field first causes the stream of electrons to bend sharply to the left side of the screen and then steadily sweeps it across towards the right, leaving a single, perfectly straight glowing "scan line" across the screen. The beam is then caused to snap back to the left, but pointed a tiny bit lower, so the next sweep to the right draws a second perfectly straight scan line across the screen, but a little lower than the last one.

This process repeats again and again, steadily approaching the bottom of the screen. But as the lowest possible scan line is drawn, the beam is snapped back up to the top left corner. From there it begins again, sweeping out one scan line after another, gradually filling the screen with lines, each laid down exactly on top of the set just completed.

In nearly every television and monitor tube in this country, the electron beam carries out a little less than 16,000 horizontal sweeps every second. Even more chilling, during the CBS evening news, when millions of TV sets are tuned to a single broadcast channel, millions of these beams, all over the country, sweep from side to side in exact and perfect synchrony. In every home, the beam snaps back to the top at the same instant, 60 times every second, performs exactly 262.5 horizontal sweeps, and then snaps back up to the top again (262.5 times 60 equals 15,750 sweeps per minute).

Figure 5.1 shows some of the scan lines in the sweep pattern. This pattern is called a "raster," the electron beam is called a "cathode ray" and the events all take place inside a Cathode Ray Tube (CRT). The yoke is controlled by circuitry inside every television and every monitor. The raster pattern starts up immediately as soon as the set is turned on, whether or not it is receiving any TV signals or input from a monitor cable. However, there are two ways in which the CRT will change its standard raster behavior in response to incoming signals.

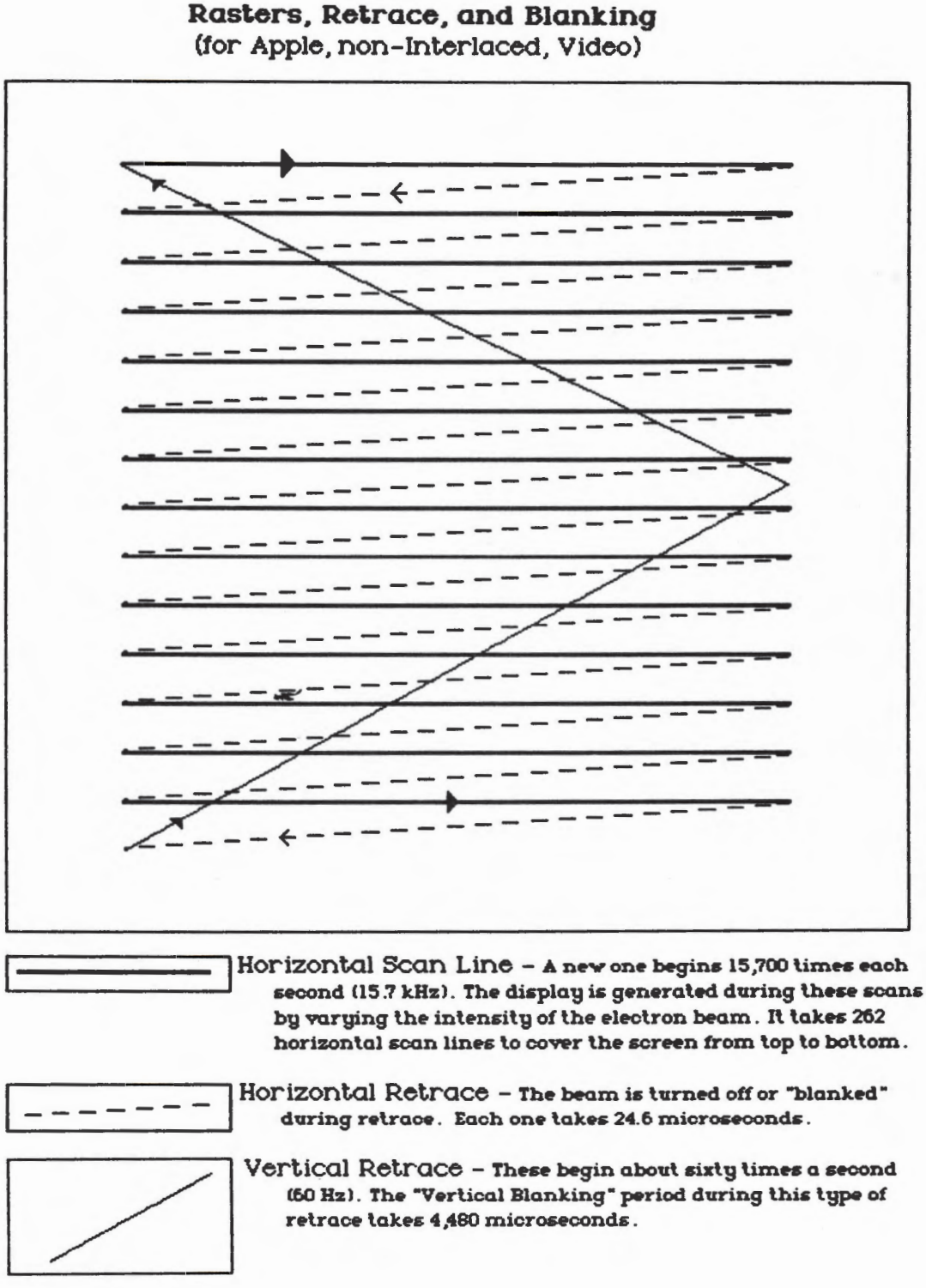


Fig. 5.1 Non-interlaced raster pattern for CRT screen.

Synchronization and Blanking

The two kinds of responses are used to synchronize the starting time of each horizontal and vertical return or "retrace," and vary the intensity of the electron beam. In a broadcast television system, the sweeping in the CRT is synchronized with the sweeping of the camera tube in the TV camera, and in the Apple it is synchronized with sweeps through a display range in Apple memory. Synchronization is accomplished by sending brief little pulses to the proper place in the monitor, one kind of pulse for horizontal synchronization, another kind of pulse for vertical synchronization. The synchronization task is very straightforward, the horizontal and vertical synch pulses are always sent in exactly the same sequence, every time, again and again without variation.

The intensity signal is controlled by two different systems in the Apple. The first is a sort of "dumb" system which performs a very simple task. Its purpose is to be sure that the beam is off during the horizontal and vertical retrace steps. No matter what the beam has been doing as it sweeps across the screen, it is always turned way down before it is snapped back to the other side or to the top, then turned on again when it is ready to return to begin the next horizontal sweep. In this way, the beam can only draw stacks of horizontal lines. The diagonal "retrace" lines (seè Figure 5.1) are always hidden, a process called "blanking."

Intensity and the Image

The rest of the video system in the Apple is concerned with controlling the intensity of the beam during horizontal sweeps. If you turn on your monitor and type a line of text onto the screen you can get some idea of what the Apple is up to. On close examination you'll see that each letter is made up of a set of glowing dots. In each character, the dots are arranged in seven rows. Each row continues horizontally across the screen passing through each character in the line of text. Each of these rows is exactly one horizontal sweep of the electron beam, so this means that the electron beam must complete seven full horizontal sweeps to draw out a line of text.

As the beam travels along a row, it turns on or off in a carefully controlled pattern. The Apple's standard character set treats each letter as a pattern or matrix of seven rows with five dots in each row. Figure 5.9 shows the pattern for the letter A. With this pattern in mind you can look back at the line of text on your screen and pick out one of the seven rows. As you follow it along the screen, you can see that it is just a pattern of on and off signals. This pattern is controlled by the intensity signal and later on we'll say more about how the pattern is created.

Dots and Characters

For each Apple character there must be one dot for spacing between letters, five dots for the letter itself, and one more dot for spacing on the other side. To put 40 characters on a line, this comes to 40 by seven equals 280 dot positions. When the //e 80 column mode is used or a standard 80 column card is installed, their must be 80 by seven equals 560 dot positions. One important question is: How fast must the beam turn on and off to generate this many dots in a single sweep?

At a rate of 15,700 sweeps per second, each sweep has to get finished in just about 64 millionths of a second. The beam has to turn on and off 560 times during one 64 millionth of a second!

Actually, the situation is even worse because in Apple video the beam spends about one third of its time either off the screen or doing retrace. When everything is calculated out, it is revealed that the beam must be able to turn on and off in something less than one tenth of one millionth of a second.

The standard way of stating this is as a rate of about 14 million times a second (14 megahertz). The exact number is 14.31818 MHz, and some of you will no doubt remember from the back corner of you mind that this is the rate of the Apple's master crystal (see Figure 5.2). We see, therefore, that nearly all timing of events in the Apple is tied to the need to turn dots on and off very rapidly on the screen. This takes place 14 times faster than the 6502 can operate, so most of the data manipulations and calculations are performed by the Apple's Video Display Generator, almost independently of the 6502.

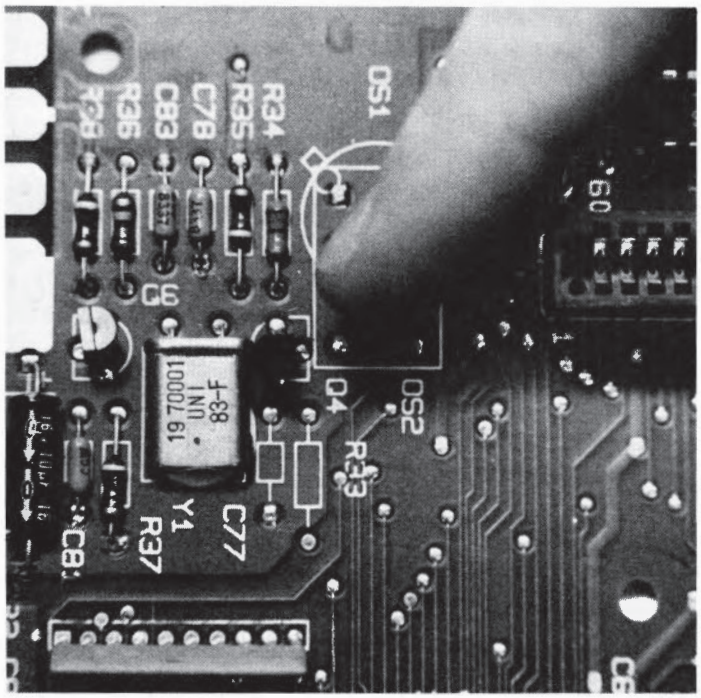


Fig. 5.2 Master timing crystal for the //e. This small metal package contains a flake of quartz which vibrates at the precise rate of 14.31818 million cycles per second.

The Apple Video Output Signal

Although there are several kinds of information the Apple must communicate to the video monitor, there is only one wire which connects them. Inside the Apple, the horizontal and the vertical synch signals, the blanking signals, and the intensity signals are all brought together and mixed to form a single "composite" signal.

If you purchase a video monitor for your Apple, you must make sure that the monitor accepts a composite signal. Some other microcomputers, such as the IBM PC provide three separate signal lines for the two synchs and the intensity, all using the five volt signals so dear to TTL electronic components (see Chapter 12). The Apple's composite video signal is always between zero and one volt, so it won't work with these "TTL" monitors.

Use of a composite video signal is a tradition from television. For broadcast TV, this composite signal was then mixed again with a high speed radio frequency (RF) signal and sent out onto the airways. At a receiving television set, the RF signals was removed electronically to reveal the composite video, and then the composite video was disassembled to get out the synch and intensity signals for use in controlling the electron beam. Because the Apple II generates a composite video signal, it is a simple manner to add an "RF modulator" which creates the kind of signal that is broadcast by TV stations. This trick permits you to connect the output of an Apple directly to the antenna of a standard TV set. A good quality RF modulator is the Sup'R Mod from M & R Enterprises, and you get one free from Apple if you buy a //c.

Television Set versus Video Monitor

This is a neat trick and it has saved many people the cost of a special monitor, but there is one serious drawback to using a TV set. In television land, the rule is that you never put more than about 300 dots on a scan line. Therefore, most television sets have been designed with electron beams which can turn on and off at a rate or "bandwidth" of only six or seven MHz (see discussion of rates above).

This is fine for the Apple's standard 40 column text, but if you use the //e 80 column card or any other 80 column card, the TV's electron beam won't be able to turn on and off fast enough. Remember, for 80 columns you need at least 560 dots in a scan line, and that means 14 MHz. For this reason, any Apple owner who wants to see more than 40 columns on a line must buy a video monitor with what is called a "bandwidth" of about 14 MHz. You can get by with a 12 MHz monitor, but the individual dots blend together a bit.

The 80/40 switch on the //c doesn't actually affect the video system at all. Rather, it is a simple way of warning your software that you are using a television set rather than a high resolution video monitor. When the switch is pushed down, a program can detect a "flag" in one of the Apple's $C0 locations ($C060, 49,248—see Chapter 26). The program can respond by setting up the Apple to generate 40 column text rather than 80 column text.

Interlace and High Resolution Characters

Now that we've dealt with the question of how many dots you can put on a scan line, it is appropriate to ask about how many scan lines you can fit on a screen or "field." The design of most TVs and monitors reflect the old standard from television which comes in at 262.5. The Apple actually uses only 262 scan lines, of which a few are lost off the top of the screen and few off the bottom, leaving 192 scan lines. Each Apple screen character is made of seven rows with one extra row left for spacing, so this allows for 192 divided by eight equals 24 rows of text.

Advanced Logic Systems sells an enhanced text card for the Apple called the Smarterm I version 2.0 which can generate characters in a 9 by 11 matrix. By fiddling a bit with the synch signals, they are able to use 24 by 11 equals 264 scan lines. These are very nicely formed characters, but not all video monitors are capable of generating this resolution.

Videx has just upped the ante dramatically by releasing their new Ultraterm board. The Ultraterm will be discussed in some detail later in the chapter, but at this point it is interesting to note that it can generate 48 rows of text using 12 scan lines for each character. This means 48 by 12 equals 576 scan lines on the screen. This high line density is not achieved by minor fiddling with the synch signal, but rather by using a slightly different raster pattern called an "interlaced" raster.

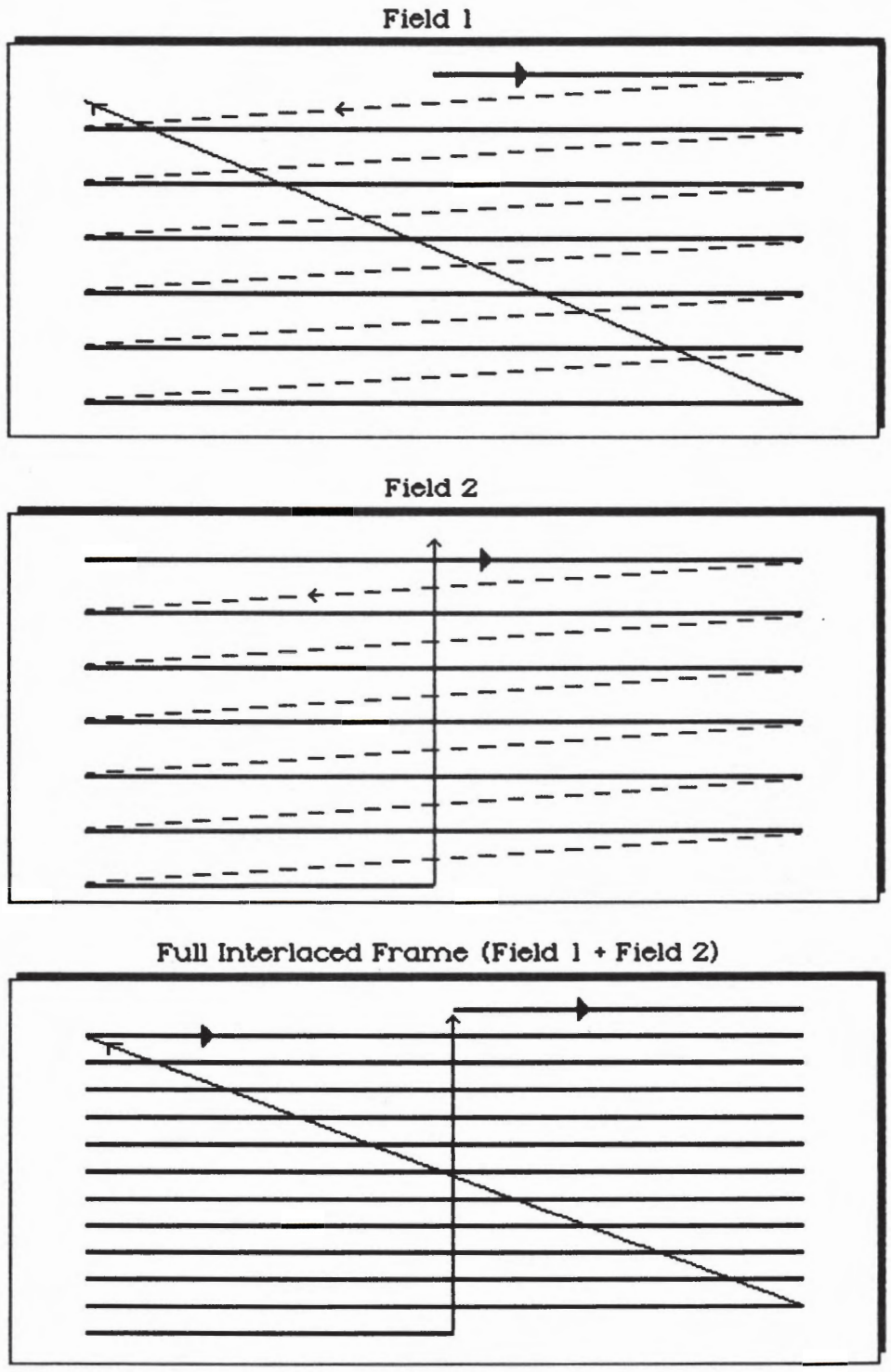


Fig. 5.3 Interlaced raster pattern for the CRT screen. This system packs in twice the vertical resolution of the non-interlaced scan. It is used for broadcast TV and by the Ultraterm video board for the Apple.

Although interlace is actually the standard raster pattern for broadcast television, it is rarely used in computer video monitors. Figure 5.3 shows an interlaced raster pattern, and you should compare it with the non-interlaced raster in Figure 5.1. In this pattern, once the 262 scan lines have been drawn, a 263rd scan line starts, but, halfway through, the vertical synch arrives and the rest of current horizontal sweep is carried out on the top. The beam now proceeds to sweep out another 262 and a half scan lines, but these scan lines are in a slightly different position than the previous set. Notice that this second time around, the vertical synch occurs at the left margin instead of in the center.

There are two consequences of this alternating vertical synch position. The first is that in standard interlace mode video there are 525 horizontal scan lines instead of 262.5. The second consequence is that each position is "refreshed" by the beam only half as often. In the "non-interlaced" mode described earlier in this section, each dot gets a fresh pulse of electrons 60 times a second, but in interlace, two full screen scans must be completed so each dot gets refreshed only 30 times a second.

In nearly every video monitor on the market, the phosphor at a given dot will have begun to fade before the beam gets back to it in interlace mode. This results in a visible "flicker" of the screen which is distracting and causes eyestrain. Video monitors have been designed with these "short persistence" phosphors so that the image will respond quickly to changes. However, this makes them dependent on a regular refresh 60 times a second.

Selecting a Monitor

The Ultraterm text card probably heralds the arrival of a new generation of very high resolution text video boards and all of these boards will need to use interlace. Fortunately, two companies offer monitors with "long persistence" phosphors which do not flicker in interlace mode. The monitors are the Apple Monitor III (see Figure 5.4), and the Amdek 300 or Amdek 300A.

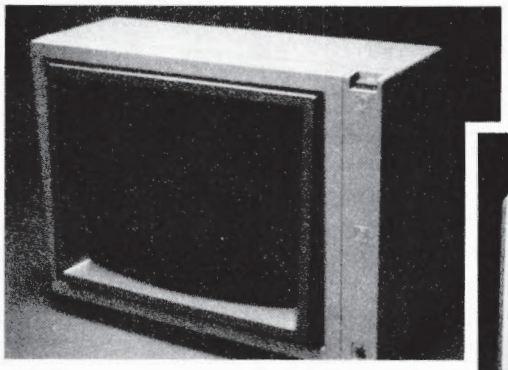


Fig. 5.4a The Apple Monitor II is a high resolution green screen monitor with fast phosphors.

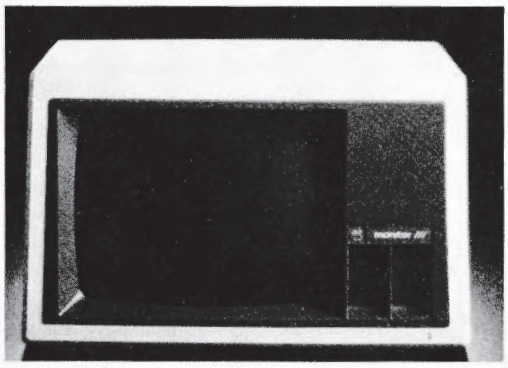


Fig. 5.4b The Apple Monitor III has slower phosphors which is desirable if you are using interlaced video.

Of these three monitors, the Amdek 300A is the outstanding video monitor on the market for several reasons. First, the long persistence phosphor makes it amenable to text generators with twice the density of horizontal scan lines as most other monitors. Second, the Amdek 300A has a relatively high resolution bandwidth of 18 MHz as opposed to 15 MHz for the Apple Monitor III. This means that on each of its nearly 600 scan lines a beam can plot nearly 1000 dots, a substantially higher resolution than the Apple Monitor III. The folks at Videx tested the Amdek 300A and discovered its bandwidth is actually quite a bit higher than the 18 MHz listed in the Amdek's product specification sheets.

The actual bandwidth of the 300A proved to be sufficient to display the 1,440 dots used in the Ultraterm's 160 column mode, and the manufacturers of the Ultraterm board have endorsed the Amdek 300A as the optimal monitor.

The Monitor III also performs better than its specifications suggest it should. However, the Monitor III can't show all 160 columns of Ultraterm text because you can't squeeze the horizontal width of the image sufficiently to bring the entire line of text into view. The USI Pi 3 has a higher bandwidth (20 MHz), long persistence phosphor, and a simple switch for displaying text in inverse, but it lacks the anti-glare screen on the Amdek 300A and Monitor III. This anti-glare feature gives the Amdek 300A a deep black background behind bright characters.

In shopping for a monitor, you should be aware that some Monitor IIIs don't have the long persistence phosphor. In some applications, long persistence leads to "smear" on the screen so Apple began producing them with a different phosphor. Then when the Apple III+ was given interlace capabilities, Apple switched back to using the long persistence phosphor on the Monitor III. The Monitor II has a short persistence phosphor but a bandwidth of 18 MHz. For any of the high resolution video generator cards, the best way to find the right monitor may be to bring the board along with you when you go shopping.

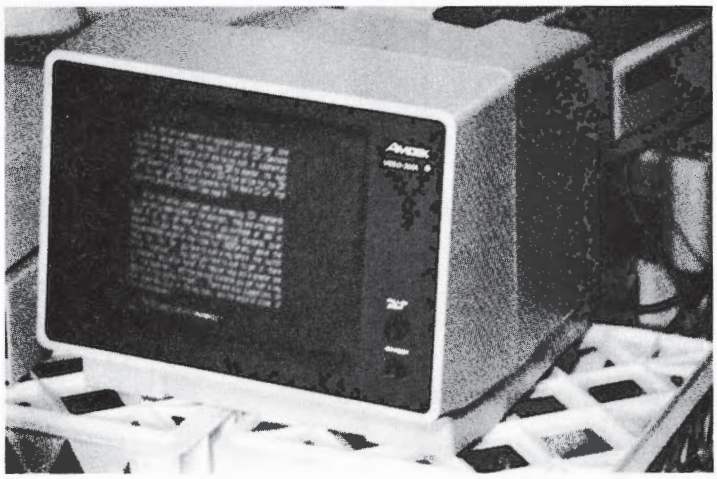


Fig. 5.5 Amdek's Video 300A has slow amber phosphors and a very high resolution video beam.

Amber versus Green

One final reason for recommending the Amdek 300A (see Figure 5.5) over the Apple Monitor III is that the 300A uses amber colored phosphors instead of green phosphors. In the early years of "monochrome" monitors the standard was black and white. Later, monitor manufac-

turers realized they could achieve a brighter image with higher contrast by using green phosphors. However, a problem has cropped up with green phosphors. The eye reacts to the green on black in some ways like it does to the classic optical illusions using green on red backgrounds—the eye muscles constantly go through tiny little refocusing actions which contribute to eye strain.

The other problem with a green screen has to do with brightness. The color receptors in the human eye are more sensitive to some colors than to others. If you place a green screen and an amber screen side by side, and use objective light measuring equipment to adjust them to the identical real intensity, the human eye will perceive the amber screen as much brighter. As a result, by judicious use of the brightness and contrast controls on an amber monitor, you can produce an image which is much easier on the eyes than the same image on a green screen. The occupational health and safety laws of several European countries actually require amber monitors.

Keeping Track of the Beam

The Apple's video display generator has a sort of mental image of what the beam is doing in the CRT. Each horizontal scan line is treated as if it were divided into 65 screen locations. Of these, 40 are considered as positions for characters, and the remaining 25 are used up during the time when the beam is off the edge of the screen or doing horizontal retrace. Each scan line is given a number from 0 to 261, so there are 262 by 65 equals 17,030 screen addresses.

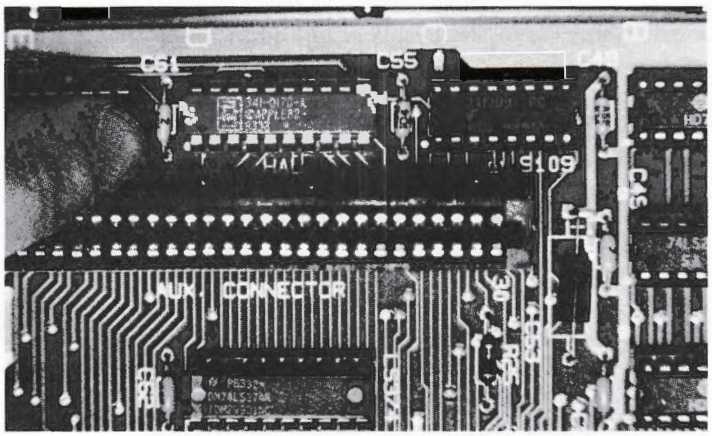


Fig. 5.6 The //e's Programmed Array Logic (PAL) chip uses an input from the master crystal to generate several special video and memory timing signals. It is essentially identical to the TMG chip in the //c.

These screen addresses are generated by a simple counting process. The master 14 MHz clock signal is counted by several chips in the II/II+, or by a special Programmed Array Logic (PAL) chip in the //e and //c (see Figure 5.6), and a series of slower clock pulses are created, some at seven MHz, some at 3.5 MHz, some at two MHz and the slowest at about one MHz. There are actually more than one of these one MHz signals, all slightly out of phase with each other, and each used for a different purpose. One of the one MHz signals is called "Phase 0," and it is used by the 6502 as its main system clock. Another of these one MHz signals, called LDPS, is used by the video screen address generator.

The screen address generator counts the incoming pulses to determine which one of the 65 horizontal positions is being illuminated by the electron beam at any instant. Each position

is valid for a full one MHz clock cycle, and then the counter is incremented. As each LDPS pulse comes in, the counter increases its current total, and announces the number to various other chips. When the 65th pulse arrives, it clears itself back to zero and causes several other things to happen.

(Afficianados may want to know that this 65th pulse is a doozy. Wozniak used another fudge here to handle his color synchronization problems. At every 65th pulse, the output of the 14 MHz main system clock is turned off for two 14 MHZ clock signals. As a result, even for the 6502, you get 64 "1 MHz" clock signals which are 978 nanoseconds in length followed by a 65th pulse which is extended to 1117 nanoseconds in length. The clock rate usually given for the 6502 is 1.020484 MHz, but this is just an average. Hardware designers who have critical timing demands should learn more about this—see Winston Gayler's book *The Apple II Circuit Description* or Jim Sather's *Understanding the Apple II.)*

Pulse number 65 signifies the end of a scan line, so the horizontal address counter sends a signal to the vertical address counter to let it know that it's time for a new scan line. The vertical address counter keeps track of these pulses as it counts up from zero to 261. At every instant it is possible to check these two counters and find out which scan line the beam is sweeping on and where it is along the line.

But how does the electron beam know what the counter is doing? Remember the horizontal and vertical synch pulses? Well, the screen addresses in the counters are used to generate the synchronization pulses. Whenever the horizontal counter gets to the 65th count, it sends a horizontal synch signal to the CRT. Similarly, whenever the vertical address counter gets to the 262nd scan line, it sends out a vertical synch signal.

Using the Screen Addresses

You may not have made it through the preceeding three or four paragraphs, but the result of all that is that the Apple assigns a number to every position on the screen which can hold a text character, and it knows when the electron beam is passing through each of the positions. All this knowledge is used to control the vertical and horizontal synchronization pulses, and to blank out the signal during horizontal and vertical retrace.

The next task is for the video display generator to find out which character it is supposed to be displaying in each of the positions. This brings us to "display memory." In the Apple there is a range of memory which is always used to store characters for display. In this range, which is called text page one, there is space for 40 characters in each of 24 lines of text. There are a variety of ways of managing screen memory, but the Apple uses a very straightforward arrangement. Each of the $40 \times 24 = 960$ positions in display memory is assigned to one of 960 positions on the surface of the screen.

When the 6502 wants a particular character to be displayed in a particular position on the screen, it puts the ASCII code for that character into the appropriate position in display memory. After that, it is the responsibility of the video display generator to take care of all the details of actually getting it into the appropriate position on the screen.

The Memory Mapper

The video display generator includes a set of chips which together are called the memory mapper. This set of chips, along with the address generator system, has disappeared into the IOU in the Apple //e and //c (see Figure 5.7), but memory mapping still goes on. This system

reads in each horizontal and vertical electron beam position from counters and decides which character in display memory to fetch.

On the first video scan line there are 40 positions that need characters, with the remaining 25 positions being off the edges of the screen. As the beam is about to enter each of these 40 positions, the memory mapper picks out the appropriate location in display memory and fetches the character stored there (see Chapter 21, Figure 21.6 for the actual address map).

Recall that it takes seven horizontal sweeps to make it from the top to the bottom of a character. In this first scan line only the top one seventh of each character is actually painted onto the screen. In the next scan line the same set of 40 characters are fetched again one by one, and the second seventh of the character is displayed. Only after all seven scan lines have been painted is a full line of complete characters visible, and only after an eighth blank scan line for spacing is drawn out does the memory mapper start looking for a new set of 40 characters.

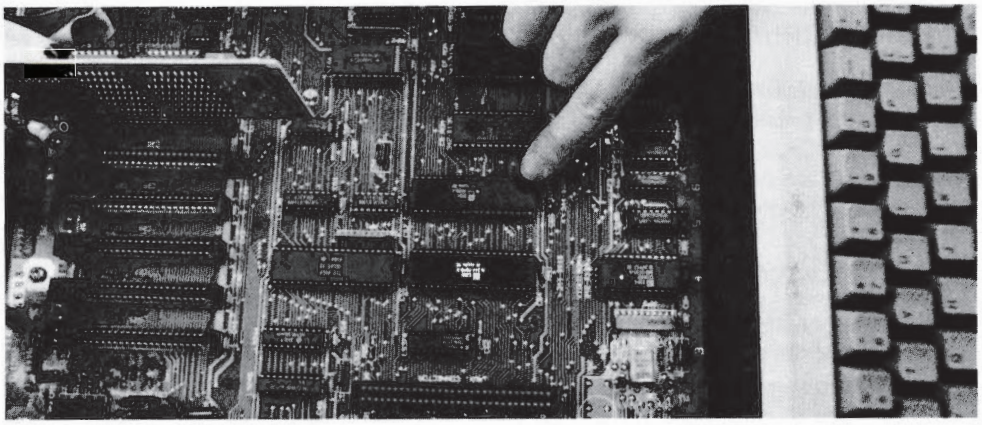


Fig. 5.7a Video address generation in the //e and //c is managed by the Input/Output Unit (IOU).

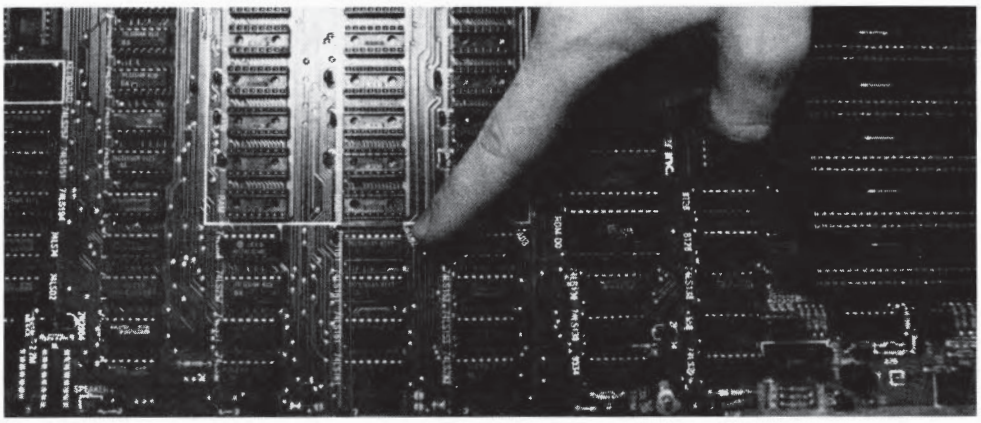


Fig. 5.7b More than a dozen chips are required in the II/II+ to do the video work of the //e or //c's PAL and IOU. The finger points to four chips in the address generation system.

Memory Contention: Who's On First?

One problem that has to be avoided in all computer video systems is the possibility that the microprocessor will be trying to change an address in memory at the same instant that the video display generator is trying to read from memory. If both try to use memory at the same time, you have to choose between confusing the microprocessor or filling the screen with messy hashes—neither very desirable. The Apple uses one of the smoothest and cleanest possible schemes for avoiding these simultaneous accesses.

The microprocessor clock signal is a square wave which is off for half a microsecond and then on for half a microsecond. The rules of memory use in the Apple are that in the first half of every microsecond (millionth of a second), when the Phase 0 signal is off, the video generator has full and unrestricted control of the Apple's address bus, data bus, and memory chips. It must complete its business and get out of the way during the second half of every microsecond when the Phase 0 clock signal is on. This second half is reserved entirely for the use of the 6502. In this fashion, the 6502 and the VDG are completely interleaved at all times (see Figure 5.9).

Translating the ASCII Codes into Video Dots

What the memory mapper comes up with for each screen position is an ASCII character code, but this is not the same as a pattern of light and dark on the screen. To get the actual pattern of dots, the video system uses a ROM chip called a Character Generator (see Figure 5.8). The ROM contains the appropriate on and off signals for each scan line of each character. When an ASCII code arrives at the ROM, the appropriate dot pattern for that character is called up. The ROM then checks to see which scan line is about to be drawn and picks the appropriate one seventh of the character. This process is diagrammed in Figure 5.9.

The appropriate set of seven dots has now been identified and these are popped out one after another at a rate of seven MHz to turn the intensity signal on or off. This is the final step in causing the right dot to appear in the right place in each character on the screen.

Changing the //e and //c Characters

It is possible to change the character generator ROM so that it will respond to each ASCII code with a completely different dot pattern. For instance, you could plug in a sanskrit character set and each ASCII code would produce a sanskrit character instead of an American character. The //e and //c are built to take special advantage of this. The //e and //c character generator ROMs contain several different character sets, two of which are easily interchanged. This feature is used to simulate an Apple II+, but it can also be used for a variety of European and other than American character sets (the English one would have a pound sign instead of a dollar sign).

The "keyboard" switch on the //c can simultaneously change the codes coming from the keyboard and select the appropriate section of the character generator ROM to accommodate any necessary European or other foreign language characters. In fact, if you look closely at Figure 5.8c, you can see that there is room for a larger ROM chip in the same built-in DIP socket, thus allowing even more character output flexibility in special versions of the //c.

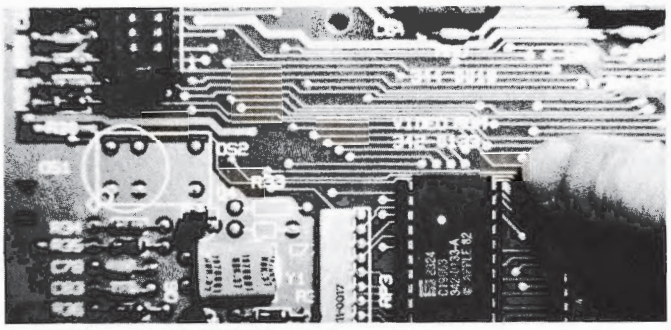
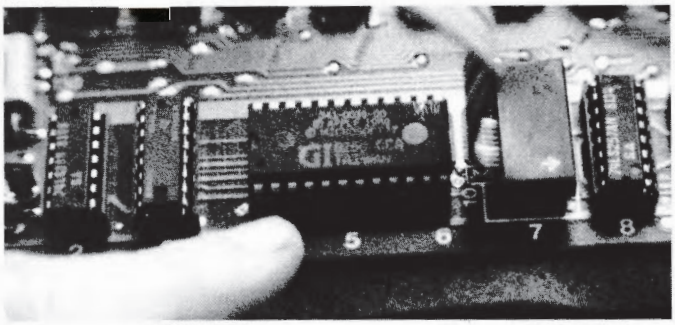


Fig. 5.8a (top) The //e video character ROM.

Fig. 5.8b (center) The II/II+ video character ROM.



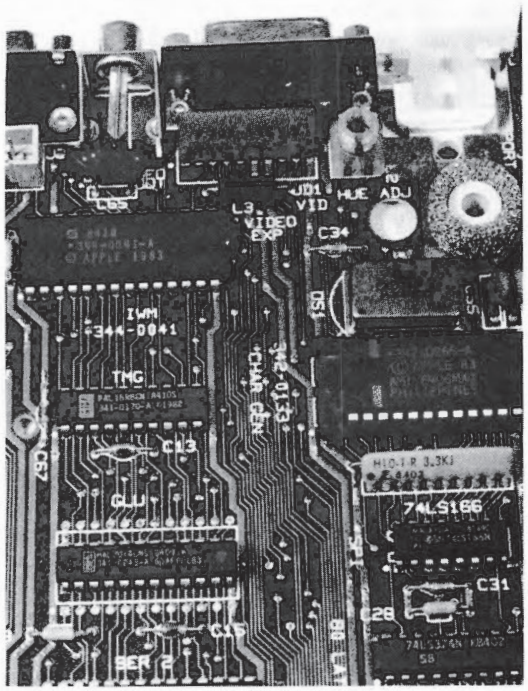


Fig. 5.8c (bottom) Most of the //c video hardware is in a small area of the motherboard just behind the video expansion connector. Here you can see the 14.31818 MHz crystal; the TMG timing chip; the character generator ROM that contains the MouseText Icons as well as standard characters; the 74LS 166 parallel /serial shift register that converts a byte of video data into a stream of bits for output; and the VID hybrid amplifier.

Lowercase Character ROMs for the II and II+

Apple II and II+ owners can purchase a replacement for the standard II+ character generator ROM which will permit their machines to generate lowercase characters when operating in standard Apple 40 column mode. These ROMs usually substitute lowercase characters for flashing or inverse uppercase characters. The most popular of these is the Dan Paymar LCA-1 and LCA-2. The LCA-2 is for revision 7 and more recent Apples, while the LCA-1 is for older Apples (see Appendix D to find out which kind you own). These lowercase ROMs are also sold by Lazer systems, by Vista (Vision-20), and by MPC Peripherals. They are very handy to use if you are programming in BASIC or Assembly Language, but most commercial software will not take advantage of this kind of lowercase characters. Vista also sells a card called the Vision-40 which permits a programmer to create a variety of special fonts for other languages or for special math and science applications.

Scrolling

In the Apple II, scrolling is handled by the 6502 and the Apple's built-in I/O machine language routines. All of the work that produces scrolling goes on during the 6502's half of each microsecond and requires absolutely no attention from the video display generator.

The Apple's scrolling system is based on the strict assignment of each location in text display memory to one specific and permanent character position on the screen. When a row of text characters has been stored in the third row of display memory, it will get painted on the third row of character positions on the screen. To move that set of characters up to the second row, the 6502 uses a subroutine which copies the bytes from the display memory third row to the display memory second row, and then erases them from the third row in display memory. It takes about 10 or 15 millionths of a second (microseconds) to move each character, so the whole line gets scrolled in about 400 microseconds. In a full page scroll, the top line is erased and overwritten, and the bottom line is cleared for new text.

When the whole screen is full of characters and all 24 rows of text have to be changed, the 960 one byte memory moves take the 6502 a total of about 10 or 15 milliseconds. The 6502 pretty much shuts down all other functions and does nothing but scrolling during this block of time. It accepts no new keypresses, does no other work. This may seem to be fairly fast, but if you have data coming into your computer from a 1200 baud modem, a new character arrives every 12 milliseconds and the scroll routine is just barely fast enough to finish its work and snap back to attention in time to catch the next incoming character after the end of a line.

Review of Apple Text Video Generation

The electron beam in the CRT sweeps out 262 horizontal scan lines in each field. The speed and pattern of these sweeps is determined largely by national standards for television. The Apple has been built to generate its video display at a speed very close to the standard CRT sweep rate, and it is able to force the CRT into exact synchrony with its own circuitry by sending out a horizontal synchronization pulse when it wants each scan line to end, and by sending the beam back to the top of the screen with a vertical synchronization pulse at the end of each field.

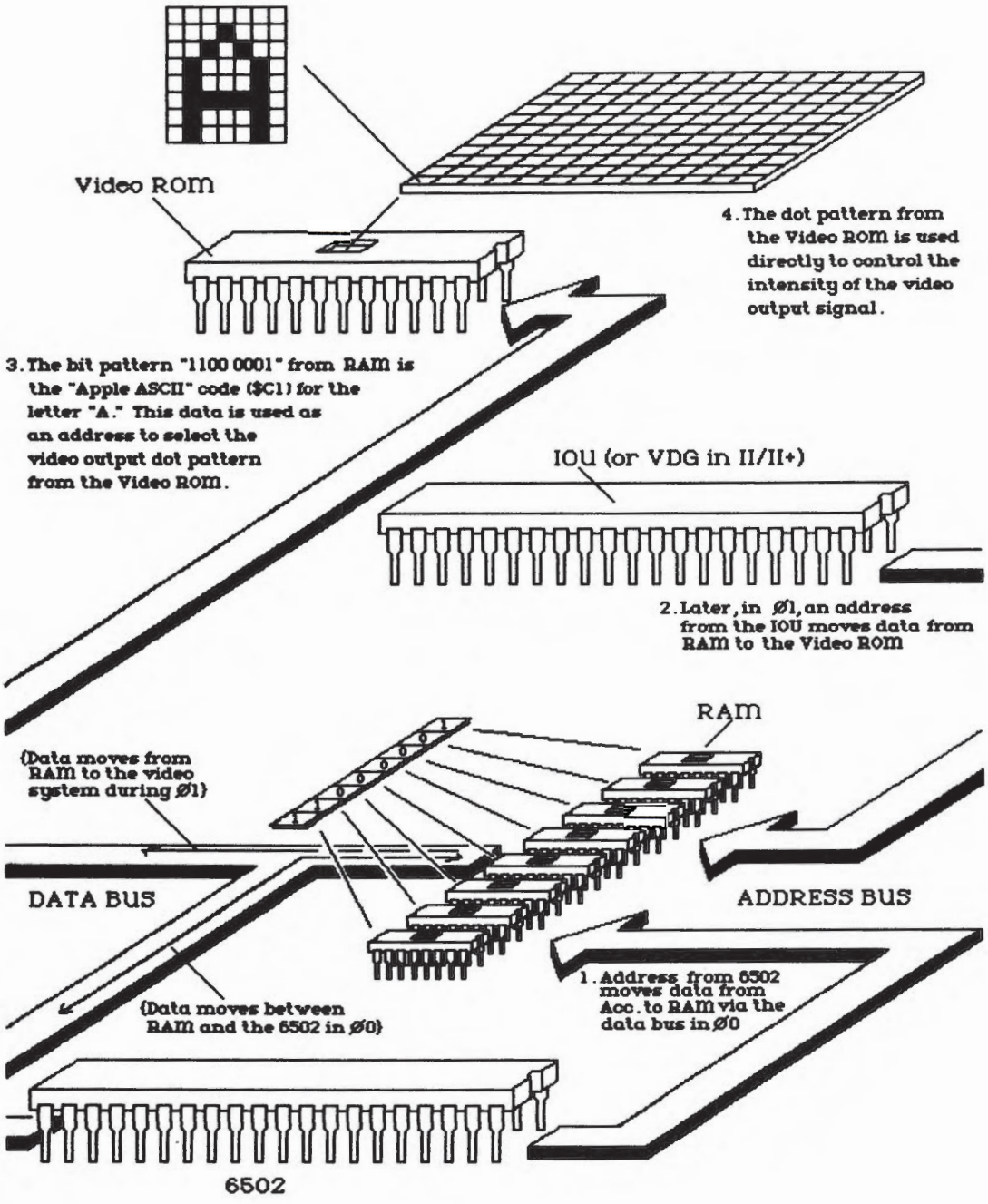


Fig. 5.9 A master overview of the 40 column text video display generation system.

Of the 262 scan lines, 192 are actually visible on the screen, and since each character is eight scan lines tall, the displayable video scan lines are grouped into 192 divided by eight equals 24 rows of text characters. In each of the 24 rows, the Apple assigns 40 positions for characters, so there are a total of 24 rows by 40 columns equals 960 character positions on the screen.

Each of the 960 character positions on the video screen is paired with one of 960 locations in the Apple's motherboard RAM. This area of display memory RAM is called text page one. To display a character, the 6502 places an ASCII code in the appropriate position in screen memory. The 6502 and the person programming it can cause characters to move around on the CRT screen by actually moving them around within display memory RAM. Scrolling is accomplished in this way by recopying every character in the display memory to a different location within display memory.

The video display generator scans the display memory in synchrony with the electron beam's passage through the screen positions. For each position on the screen, the appropriate character is fetched from the appropriate location in display memory. A character generator ROM chip contains the actual pattern of dots which will form the character on the screen. The dot patterns are read out of the ROM and sent out to the monitor to dim or brighten the electron beam at a rate of seven MHz. The complete screenful of characters is drawn over and over again 60 times a second to prevent the glowing phosphors from dimming.

Flat Screens

One very important display option for //c users is to not use video at all. The Apple flat screen Liquid Crystal Display (LCD) plugs into the video expansion connector and gives you most of the features of a CRT without all the weight and heat.

Everything described above in relation to the video generation system still applies. However, once the signal emerges from the back of the Apple, a number of things change. The LCD screen contains a custom designed chip somewhat like the IOU and MMU. It is responsible for supervising the translation of the video output signal from the Apple into LCD control signals.

There are two tasks involved. First, there is a "frame grabber" which captures the incoming video signal and converts it back into a bit pattern in RAM. That's right, all it does is to strip out the synch and blanking information and recreate what the Apple has stored in its display memory RAM. For this purpose, it is provided with 16K of its own RAM so that it has enough space to store one complete 80 column screen or super high res 560 by 192 graphics screen.

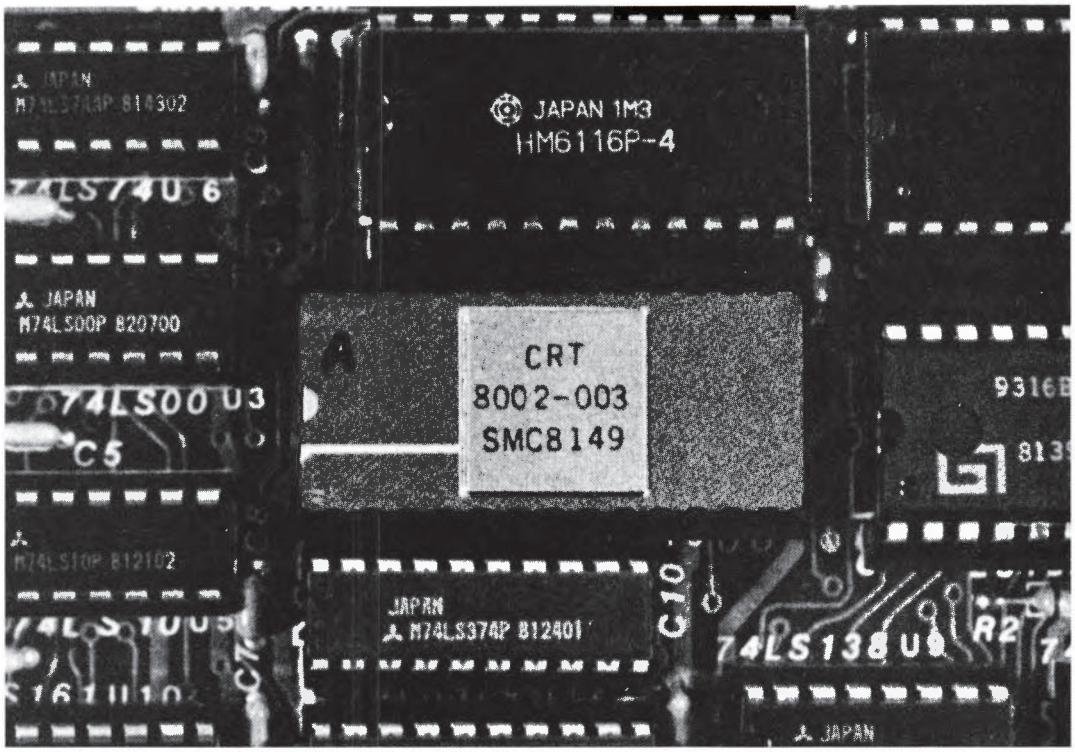
There is another section of the custom chip which acts as an LCD Frame Refresher. Its task is to read the data stored in the 16K of RAM and use the information to activate the appropriate LCD crystals. Its task is very different from a CRT electron beam. There is no rastar. Rather, it's simply a matter of activating the appropriate locations on the screen grid by turning on what amounts to an X line and a Y line.

The two parts of the custom chip (the Frame Grabber and the Frame Refresher) do their work interleaved, at the same time. This is in many ways like the relation of the IOU and the 6502/ MMU on the Apple motherboard. There is a division of labor into two different phases of a clock.

The master clock for all this is the same 14 MHz clock that operates the Apple's motherboard. This clock signal is carried out from the video expansion connector to the LCD display. There is a potential problem here. Signals which operate at such a high speed are very sensitive to being placed in long cables. If the signal was carried directly out of the Apple, the length of cable would provide "parasitic capacitance" (see Chapter 13), which would have a disastrous effect on the whole Apple. Thus, there is a little buffer chip which detects the 14 MHz signal from the video expansion connector and then reproduces it, in isolation, in the cable. This chip is responsible for the bulge in the cable near the connector.

The LCD screen itself acts in a somewhat different a manner than a CRT because of differences between the properties of the liquid crystals as opposed to the glowing phosphors in a CRT. The most important difference is that a phosphor begins to glow brightly almost immediately after it is stimulated by the CRT electron gun. A liquid crystal, however, takes as long as 100 milliseconds to reach full intensity. This is slower than the 60 Hz refresh time used to update the CRT monitor. What all this means is that you can move, for instance, a mouse cursor from place to place on a CRT screen with a position change 60 times a second, but if you try that on an LCD, the cursor will just disappear. An LCD dot has to be selected during six full VBL refresh cycles to reach full intensity.

The other important difference has to do with the shape of the dots. The 560 by 192 dots are each perfectly square. Therefore, the screen is a rectangle about twice as wide as it is tall. On the video screen however, the same dots fit into a screen that is approximately square. This means that there will be considerable distortion of the shape of a graphics image if it is designed on a CRT but dispayed on an LCD screen.


JAPAN 1M3
HM6116P-4
CRT
8002-003
SMC8149
JAPAN
M74LS00P 820700
74LS00 U3
C5
JAPAN
M74LS374P 812401
C10
74LS138 U9
R2
JAPAN

Chapter 6

80 Column Text Display

80 Column Cards

Apple chose to use a 40 column screen display because it could be drawn on a video screen by blinking the electron beam on and off no faster than seven MHz. At this rate, Apple video would work with the television sets most people already owned. However, the industry standard in computer video is an 80 column display.

Eighty columns are standard for two reasons. First and foremost, data processing had always been done on IBM punched cards (Remember "Do not fold, spindle, or mutilate"?) which had 80 characters per card. CRT terminals were originally set up to emulate and replace these punched cards.

The other reason for 80 columns is much more important for most microcomputer users, and that is the need for more than 40 columns when word processing. Actually, most typed material on paper has traditionally been about 65 or 70 columns wide. Forty was not enough, and 80 was the data processing standard, so 80 column video became the microcomputer standard as well.

Owners of Apple II and II+ computers often bought a card which carried the complete guts of a stand-alone CRT terminal in order to generate 80 column text. These cards include what amounts to a serial port, their own display memory, and a complete video display generator system. The most popular of these 80 column cards has been the Videoterm card made by Videx.

The Apple //e and //c use a much simpler approach to generate 80 columns of characters. The //e and //c 80 column systems are fully integrated into the Apple's older video display generator, so it is relatively easy to control from Applesoft BASIC and by using the machine language I/O subroutines built into the Apple. However, the additional effort of generating 80 columns does sort of strain the capabilities of the Apple's standard video generator and its display memory software.

There is nothing sacred about 80 columns, however. Punched cards are not very important these days and never had much to do with microcomputers anyway. And so, the folks at Videx said "Why not 160 columns for VisiCalc?" Also, there is nothing sacred about 24 rows, so the folks at Videx said "Why not 48 rows?"—and so the Ultraterm card was born. This is one of the finest quality text video generators available for any computer or terminal system. It is very similar to the Videx Videoterm from the point of view of a programmer, but it is able to put twice as many characters onto the screen in any one of several different combinations of row and column formats.

The //e and //c 80 Column Text System

The //e 80 column system requires little more than a 1K RAM chip and a few simple interface chips on a card in the auxiliary slot (see Figure 6.1). The machine language software which manages the 80 column system is actually in a ROM chip on the Apple's motherboard and it can be used to improve some features of the 40 column system, even if you don't buy the //e 80 column card. This also means that an auxiliary slot card from another company will function exactly like Apple's own 80 column card since it will be operated by the Apple ROM on the motherboard. The built-in 80 column system on the //c motherboard has exactly the same chips that you would get on a fancy 64K //e extended text card.

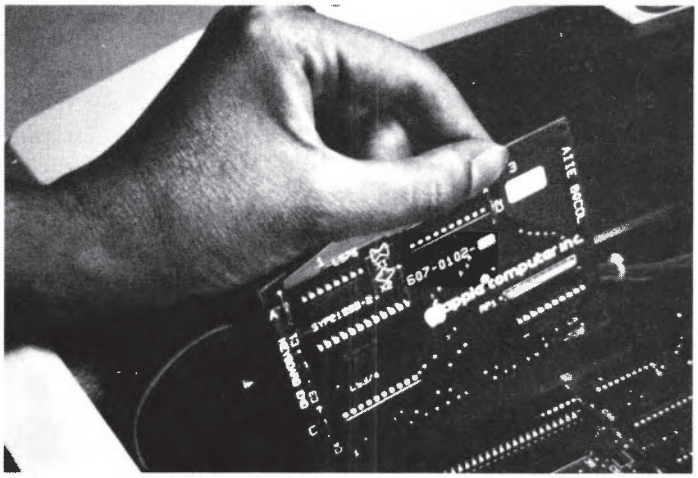


Fig. 6.1 The //e text card. In selecting a card for the auxiliary slot, keep in mind that that the only function you may not be able to get from any other slot in the //e is a special kind of color output called RGB.

Several companies are selling //e auxiliary slot cards which provide the 1K of RAM, and also provide additional features, because the auxiliary slot has pins that carry special signals for controlling additional memory and for graphics. In selecting a card for the auxiliary slot, keep in mind that the only function you may not be able to get from any other slot in a //e is a special kind of color output called RGB (see Chapter 7). This is because an RGB card requires a particular signal from the Apple which is available on slot 7 of a II or II+, but is only available on the auxiliary slot of a //e. This limitation only applies to owners of revision A // es (see Appendix D), since these signals have been restored to slot 7 in later revisions of the //e. The signals needed to create RGB output are all available on the external video expansion connector of the //c (see Chapter 7).

//e 80 column cards which provide an additional 64K of RAM memory are available from Apple, Coex, ComX and Quadram, or you can get a 128K card from MicroMax, or better yet, one from Saturn Systems (the Neptune card) which provides 192K of additional RAM (see Chapters 24 and 26 for information on using all that RAM). Putting one of the 64K auxiliary slot cards into a //e makes its video and memory configuration identical to a //c. However, the high capacity RAM cards can make the //e far more powerful than a //c.

Taxan and Amdek make //e 80 column cards which can also generate RGB color monitor outputs (see Chapter 7) and Taxan makes one with RGB outputs and 64K of RAM. Finally, Microsoft makes a card for the auxiliary slot which provides //e 80 column output, a fast Z-80 processor, and 64K of RAM (see Chapter 29).

Operation of the //e and //c 80 Column Systems

The principal design element in the //e and //c 80 column systems is the addition of more display memory. In 40 column mode, there are 40 by 24 equals 960 character positions on the screen, so only 960 bytes of display memory are required. In 80 column mode, however, there are twice as many character positions on the screen and so the Apple has to provide twice as many memory locations for display memory.

This additional memory requirement comes to just about 1K of memory, which is not an awful lot, but Apple felt that there was no "safe" block of memory quite that big that they could use without interfering with existing Apple software. To get out of the problem, Apple chose a "bank switching" scheme in which an extra 1K memory chip was made available but in which the addresses were exactly the same as the addresses for the old text page one.

Subroutines for Odd and Even Columns

When the Apple //e or //c is loading characters into memory for display in 80 column mode, it uses a new machine language I/O routine. Machine language programmers can purchase a listing of the new program from most Apple dealers in the *Reference Manual Addendum: Monitor ROM Listings for IIe Only.* As each character arrives for display, it alternately loads one character into the special 1K RAM and then the next character into the normal text page one display memory, and so on back and forth. (Machine language programmers should check $CF2F in the 80 column ROM listing.) Major entry points are the same in the //c firmware, but most of the routines have been rewritten in 65C02 code.

When a full page has been loaded, you would see a strange pattern in the 1K RAM if you could look in. It would contain the ASCII codes for every other character you'd typed in. Of the 80 columns, the 40 even numbered columns get their characters stored in the extra 1K and the 40 odd numbered columns get their characters stored in the traditional display memory positions in main RAM.

The Video Data Bus

Now that the 6502 has created this schizophrenic situation, how does the video display generator deal with it? It is forced to break two otherwise cardinal rules of Apple operation.

The first rule it breaks is in fact a cardinal rule of all microcomputers. It reads two different memory locations at exactly the same time. Normally, in bank switching setups, it's considered OK to assign the same numerical address to two different locations but only as long as you restrict yourself to speaking to the two of them at different times. Remember, address locations can be thought of as space, and you can't have two locations in the same space at the same time. Rules like these were probably made to be broken, but the //e does have some mitigating circumstances. To wit, although they're both addressed at the same time, they are not both connected to the main data bus.

The //e and //c have a special "video data bus" which connects the character generator ROM to two 74LS 374 "octal flip-flops." When a text address is generated by the memory mapper, it causes the two locations with that address to each present their data so that it may be loaded into one of the 74LS 374s. The extra 1K is located on the //e 80 column card along with one of the 74LS 374 chips. The other 74LS 374 is located on the //e motherboard. The two are

loaded simultaneously. The one on the text card gets the character for the next even numbered column and the one on the motherboard gets the character which will go in the next odd numbered column. In the //c, all the RAM and both LS 374s are on the motherboard. Characters for even numbered columns are stored in the back half of the //c, under the disk drive, while characters for odd numbered columns are stored towards the front, under the keyboard (see Figure 6.2).

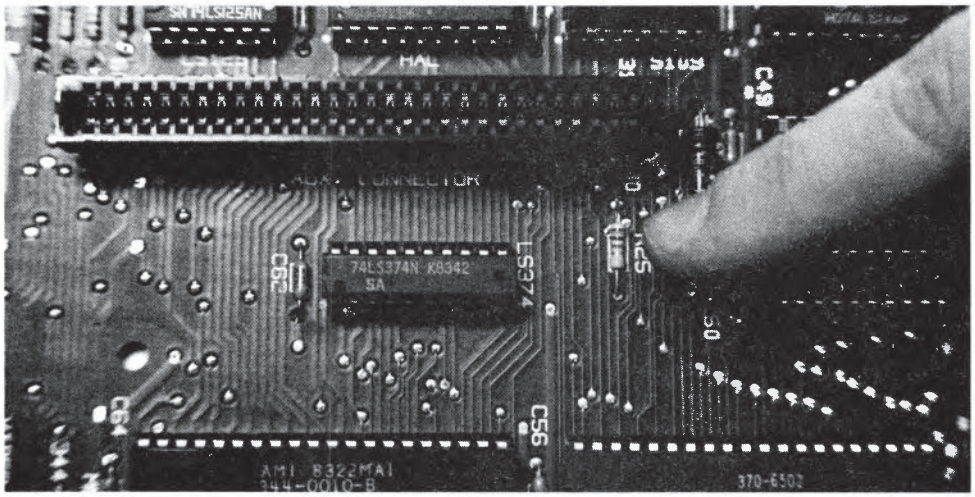


Fig. 6.2a (top) In the //e, characters for odd numbered columns visit the LS 374 chip on their way to the video character ROM.

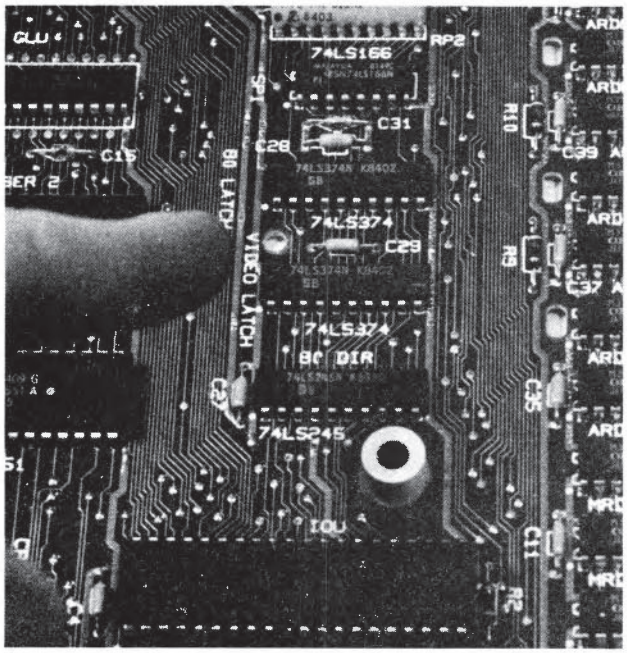


Fig. 6.2b (bottom) In the //c, characters for odd numbered columns are stored in the auxillary RAM chips (marked ARD0-7 RAM) and they go through one of the two LS 374 chips. The other LS 374 chip gets characters for even numbered columns from the main RAM chips (marked MRD0-7 RAM). The 74LS 245 chip, marked "80 DIR" on the motherboard, is involved in the storage of characters in the auxiliary RAM by the 65C02.

Timing Differences

With these two "buffers" loaded, the //e launches into the beginning of a normal II or II+ video display sequence. When the Phase 0 signal goes off, it uses its proper first half of the microsecond (see Chapter 5) to present the character code from the main memory to the character generator ROM. However, it does this via the special video data bus.

When Phase 0 turns on again, the video display generator violates another Apple law by continuing to move data even though this half of the microsecond legally belongs to the 6502. Because the video display generator in the //e has its very own data bus and a simple way of addressing the 74LS 374, it can use this second part of the microsecond. So although the II and II+ video display generator is only able to move one character in each microsecond, the //e video display generator is able to move two characters each microsecond.

There is one more trick required. In the II and II+, it takes a full microsecond for each character to get converted to the appropriate line of seven dots and get shifted out onto the intensity line at seven MHz. The //e and //c must work twice as fast to get two characters out in each microsecond, so when they are in 80 column mode they use a full 14 MHz clock to pop the dots onto the video intensity output line. In this way they are able to get all 14 dots out of the way before moving in the next pair of characters. Note also, this means the video monitor must have a 14 MHz bandwidth so television sets are definitely out of the running. You must have a video monitor to use the Apple's 80 column text system.

Scrolling the //e and //c 80 Columns

The scrolling mechanism used for the //e and //c 80 column text system is essentially identical to the approach used in standard 40 column mode (see Chapter 5), but the manipulation of the characters in the extra 1K involves the regular address and data buses, so it can't be done simultaneously with the main memory scroll operation. Rather, the 6502 first rearranges all the characters on the motherboard text page, then moves the characters into the extra 1K. The scroll operation (machine language programmers see $CCB8) therefore takes twice as long as it did with only 40 columns to move. Further, the firmware ignores any interrupt requests during the entire operation. So the system's attention is riveted on the scroll operation and you cannot distract it even if there is an urgent need for it to pause and do something else.

The II and II+ just barely finish scrolling fast enough to handle the incoming data stream from a 1200 baud modem, but the //e scroll routine doesn't quite make it. There are three solutions to this 1200 baud problem.

Solving the //e 1200 Bps Modem Problem

First, you can leave your auxiliary slot open and install one of the fast 80 column cards designed for the II or II+ (Videoterm, Smarterm, etc.). This has the disadvantage of preventing you from putting extra memory in the auxiliary slot. Most of these video cards need to go into slot 3, and this slot is disabled whenever a card is put into the auxiliary slot.

Second, you can buy an Ultraterm card. This card has special circuitry which overrides the "slot 3 killer," so you can use both the Ultraterm and extra memory in the auxiliary slot.

Third, and least expensive, you can buy new communications software which has a special capability for overcoming the problem. Fourth, and at no expense to you, it is sometimes possible to instruct the sender to send several "null" characters at the end of each line, so that the characters the //e misses will not be important.

Fifth, get the new revised ROM from Apple and see that you have appropriate software to work with the new ROM to get around the problem. This new ROM permits interrupts but still may have a speed problem.

Machine language programmers can solve the problem in the old ROM in one of two ways. Put a header on your communications software which copies the firmware into high RAM, enables interrupts during scrolling, and then sets the appropriate switches to ensure that the monitor is read from RAM and not from ROM. This approach is not without hazard since interrupts sometimes cause bizarre and erratic behavior in an older Apple II (see Chapter 27). The second way is to write a new scroll routine—this is what the Pie Writer folks have done.

Unfortunately, the only sure fire approach available now is to buy a non-Apple 80 column card which plugs into one of the regular peripheral slots, not into the auxiliary slot.

In the //c, the scroll routines are rewritten in 65C02 code, and this provides enough of a speed increase to solve the scrolling problem. But don't run out and buy a //c just because you want to use a 1200 bps modem. There is a completely different problem in the //c which has to do with a design shortcut in the baud rate generation system. This problem is explained in Chapter 17.

80 Column Cards and the Ultraterm Video Display Card

The popular 80 column video boards for the II and II+ all work by providing a complete new video display generator. Typically, they do not use the Apple's display memory, nor its clock, nor its character ROM, nor its screen addressing system. When one of these cards is activated, it intercepts all characters being typed at the keyboard or coming in from any serial port, and never even lets them get to the Apple's display memory.

These boards always include a complete set of replacement I/O subroutines stored in their own ROM (except Omnivision which requires you to load a routine from disk) which handles the placement of the characters into their own, built-in display memory RAM. The designers of these cards faced a much simpler task than Wozniak did when he designed the Apple II video system. Although many of these boards have some sort of graphics display system, none of them have anywhere near the resolution of the Apple's high res graphics. Further, none of these boards concerns itself in the slightest with the intricacies of color timing. These boards are completely devoted to generation of text displays.

Not only is the design task less ambitious, but the electrical engineering required to create a text display generator has been simplified enormously by the advent of single chip video display generators called Cathode Ray Tube Controller (CRTC) chips. The popular Videoterm and the new Ultraterm from Videx both use a chip called the Motorola 6845 CRTC. This chip is from a "family" of input/output devices built to work with the 6800 microprocessor, but which also work well with the 6502.

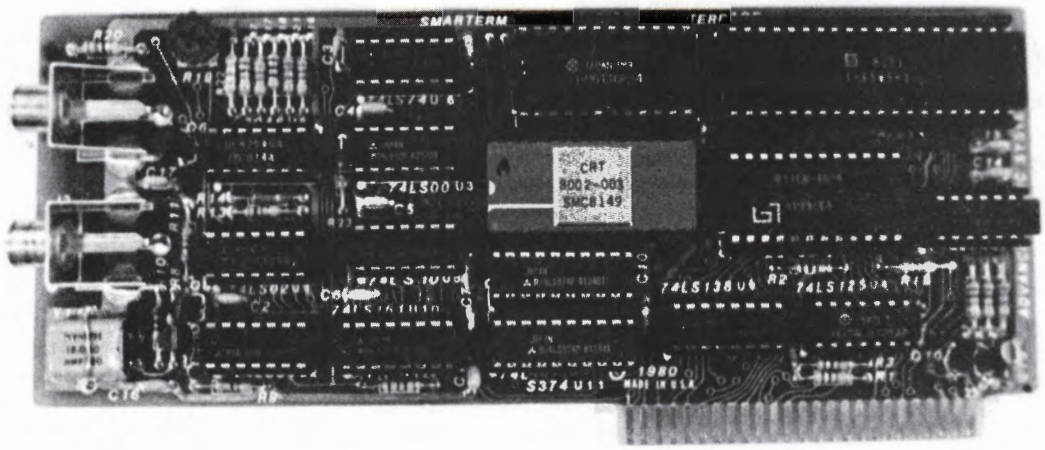


Fig. 6.3 The Smarterm I from ALS uses a 6545 CRTC (cathode ray tube controller) to manage the 80 column video display.

If you want to learn more about operating the 6845, you can read about it in the *CRT Controller Handbook* by Gerry Kane (Osborne/McGraw Hill), but the manuals from Videx are really more than sufficient. The Videx manuals deserve special mention. They represent probably the outstanding technical documentation of any Apple peripheral product. Only Apple itself does better. What this means is that professional programmers and hardware designers find it easy to design their products to work with the Videx boards so that the user doesn't have to do any special fiddling. The beginning Applesoft or assembly language programmer will also find the Videx video boards a richly documented environment in which to program. It should be noted that this openness leaves Videx open to being copied by less original manufacturers, but Videx has chosen to put the user first.

The Smarterm I 80 column board from Advanced Logic Systems (see Figure 6.3), if enhanced with the version 2.0 ROM, has more features, generates better characters, and is more versatile than the Videx board. Further, the documentation is very good for most programming tasks. The problem is that after you've invested two or three months time in building a program around the Smarterm I, you may run into some special problem or (yes) bug in their firmware. The manual is of no help when things get tricky, and ALS refuses to provide detailed information on their firmware or hardware. The Smarterm I is an outstanding 80 column card for the user of applications software and for the beginning programmer, but intermediate and advanced programmers should be wary.

The Ultraterm board from Videx (see Figure 6.4) has all the features available on any Apple video board, and although it may be out of your price range (it's $379 list, and $279 mail order, but most people will need to buy this card at the higher price from a local dealer who knows how to use it), a brief guided tour of the Ultraterm may help you evaluate the features of other boards and to appreciate why they cost less.

What the CRT Controller Does

In the course of describing the Apple II video display generator, several component tasks were identified. There must be a master crystal to provide high speed timing for the system, these clock pulses are then counted and used to keep track of when to send horizontal and vertical synch pulses to the CRT, and to keep track of where the beam is pointing. The system must also generate memory addresses to be used to call up the appropriate character to be displayed at each screen position as the electron beam sweeps along. The 6845 does all of this automatically. All it requires is an incoming clock pulse, some display memory in which to find the characters, and a character generator ROM.

In addition to these tasks performed by the Apple's video display generator, the 6845 does several important things that the Apple has to do from software. One of these is generating a cursor, and another is reading the position of a light pen. Aside from the simplicity and reliability of providing all of these functions in a single chip, the 6845 provides an important advantage in that it can be programmed to change its behavior.

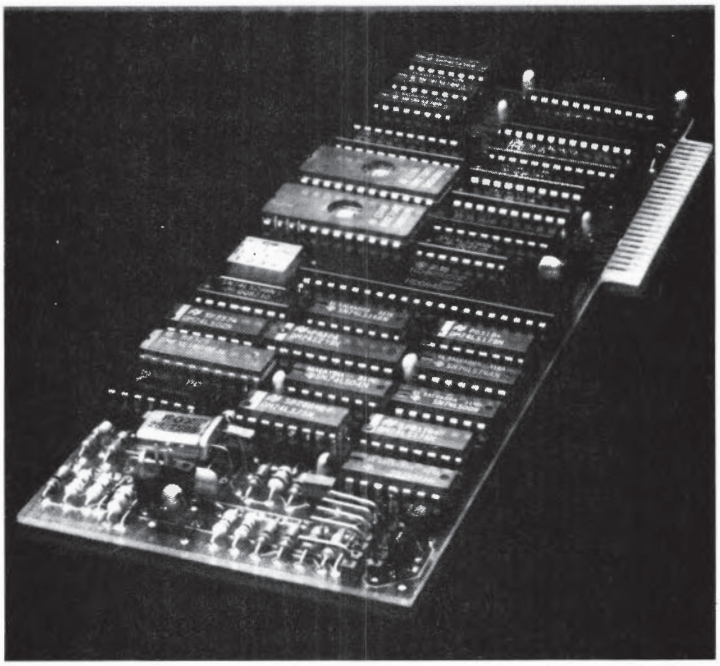


Fig. 6.4 Videx uses a 6845 CRTC and a high speed crystal to supervise their very high resolution Ultraterm board.

The Apple video display generator can be caused to change its behavior from a text generator to a graphics system, but within text mode it is fairly rigid. The Ultraterm takes advantage of a full range of programmable features of the 6845 some of which were left unused in the older Videoterm board. These include the ability to change the number of characters in a row and to change the number of rows being displayed as well as various esoterica to do with the synchronization pulses. In addition, you can program the cursor to be a single line or a full block, to flash or to stay on.

Scrolling with a CRT Controller

Another important feature handled by the CRTC is scrolling. The Apple's scroll routine is entirely software based, and requires a substantial amount of time and effort on the part of the 6502—so much so that the Apple //e 80 column card can't handle a 1200 baud modem. The 6845 handles scrolling automatically in a very simple and extremely fast way. Instead of moving the characters, it just changes its memory map. Each row of text in display memory is only assigned a temporary position on the screen. When the time comes to scroll, the 6845 just changes its matching of memory rows to screen rows. As a result, scrolling in the Ultraterm is a simple internal operation which takes only a few millionths of a second instead of a fiftieth of a second as in the //e 80 column system—an improvement of about two thousand fold.

Loading Display Memory

As a result, the limiting factor in speed of display for the Ultraterm is in the placement of characters into proper position in screen memory as they are sent to it by a program or a fast modem. This was also the case in the Videoterm board, and that board was limited to about 4800 baud (actually 6000 baud, but communications standards usually include 4800 and 9600 with nothing in between).

Someone at Videx pondered the ways of the 6502 and came up with two or three fairly subtle improvements in the firmware and management of the display memory, so that the Ultraterm is able to accept characters at 9600 baud. This may sound like more esoterica, but if you talk to someone who has used an Ultraterm, one of their principal comments will be about the remarkable speed at which the screen fills and changes. It is these few ingenious machine language changes which are responsible. When you buy an Ultraterm, part of the price you pay goes to support that programmer.

Character Formation

A very important consideration in choosing a video board is the quality of the characters which get displayed. The major factor in quality of formation is the number of dots used to make each character. The Apple's video system uses a matrix of five dots across and seven dots from top to bottom. Lowercase letters with tails such as "y" and "p" are allowed one more dot, and all other characters are allotted an eighth dot which is left blank for spacing between rows of text. Each character is also allotted a blank column of dots just before and just after it (see Chapter 5, Figure 5.9). The Apple therefore generates five by seven characters in a seven by eight dot matrix.

One of the factors in eyestrain is the density of the matrix. And although five by seven is very readable, it is not at all optimal for people who spend much time staring at characters on their monitor. Several of the 80 column cards on the market provide only five by seven characters, and may not allot an extra dot for lowercase descenders. These boards include the Sup'R Terminal from M & R enterprises, Smarterm II from ALS, Fullview 80, and Omnivision. The characters are quite adequate and at least as good as the Apple 80 column video, but certainly offer no improvement.

Several other boards offer seven by nine characters in an 8 by 12 matrix. These include the Videoterm, the Viewmax 80, the Vision-80 from Vista, the Wizard 80, and the Smarterm I from ALS. Only two boards offer even higher character resolution, the Smarterm I with the

version 2.0 ROM, which generates 9 by 11 dot characters (see Figure 6.5), and the Ultraterm which has an optional 8 by 12 dot character set in a 9 by 16 matrix.

The Smarterm I, the Vision-80, the Fullview 80 and the Ultraterm also provide an important additional feature. This is the ability to switch back and forth from Apple video to enhanced video under command from software. The other boards require you to turn off the computer and plug in your video cable differently, or reach back and flip a switch by hand, or purchase additional hardware to handle the switching. This "softswitch" capability is really essential for programmers who want to use both 80 column video and Apple graphics.

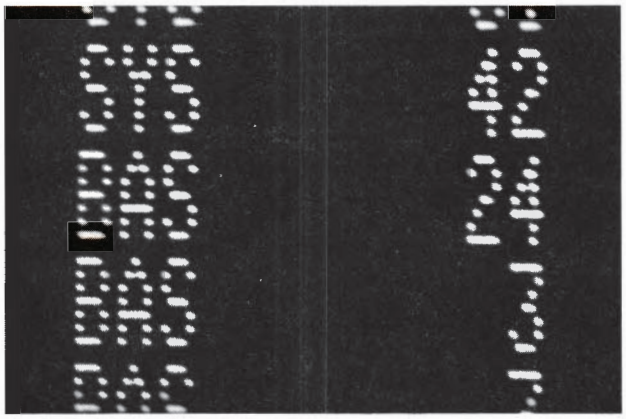


Fig. 6.5a The //e 80 column system generates 5×7 characters in a 7×8 matrix.

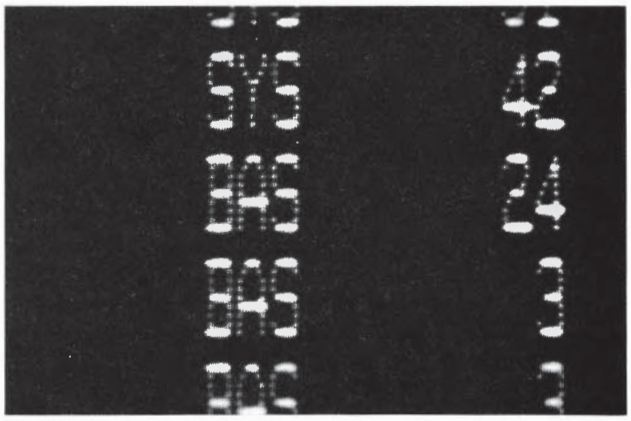


Fig. 6.5b These characters are generated by the version 2.0 upgrade of the Smarterm I. The matrix is 9×12, with lines 10 and 11 for "lowercase descenders" and line 12 reserved for the space between lines. The threaded pattern over the dots is from a silk anti-glare screen on the monitor.

Summary of Choices for Selecting an 80 Column Card

If you own an Apple //e, you can get 80 columns of text by adding what amounts to 1K of RAM on a card placed in your //e auxiliary slot. These cards are manufactured by Coex, ComX, Microsoft, Quadram, Saturn Systems (Titan Technologies), and Taxan. The advantages of this approach is that you can simultaneously add an additional 64K of memory (192K with the

Neptune Card from Saturn Systems), RGB capabilities (Taxan, see below for explanation), or a Z-80 coprocessor (Microsoft).

All of these cards use the Apple //e 80 column firmware and do nothing more than lend it 1K of RAM to do its work. This has the advantage of compatibility with all //e software and full control of 40 column versus 80 column versus graphics output from software. Also, if you own a revision B version of the //e (see Appendix D to find out which you have) and you install at least 64K, you can use super high res graphics. The three disadvantages of using the //e's built in 80 column facilities are that the character matrix is only five by seven dots, you cannot conveniently use a 1200 baud modem, and you cannot produce a simultaneous display of Apple graphics on one monitor and text on another monitor as you could with a standard video board.

Owners of Apple II or II+ computers, or //e owners who can't live with the limitations of the //e 80 column system, have a different set of choices. You can select one of several which provide seven by nine dot characters with prices ranging from $150 to $275 (Viewmax 80, Vision-80, Videoterm); but these require manual switching of the video cable to get back to 40 columns, and only the Viewmax 80 provides inverse characters (dark characters on a light background as well as light characters on a dark background). The Videoterm offers a variety of optional enhancements including extra character sets.

The Smarterm I from ALS provides seven by nine characters, with optional inverse video and the ability to switch back to Apple 40 column mode or graphics from software. It may be purchased with the version 2.0 character generator ROM to get nine by 11 dot characters. To get this many lines on the screen without interlace they had to fiddle with the synch timing, running the board near 50 Hz, and so this system is sensitive to electrical motor noise from some fans (the Kensington System Saver is one fan that does not disturb this video system).

The Ultraterm is the top-of-the-line video board. It is sold with both seven by nine and eight by 12 characters, but it is the first video board to be capable of displaying more than 80 columns or more than 24 rows. This card has several modes including 160 columns by 24 rows and, at the other extreme, 80 columns by 48 rows of text characters. It provides switching to standard Apple mode from software, and optional inverse video. It is the only video board which can operate in expansion slot number 3 while another card is in the //e auxiliary slot.

Chapter 7

Graphics and Color

Graphics

The Apple system for generating graphic displays uses the same video display generator that produces text displays. The two important differences are that there is no equivalent of the character generator ROM, and that the behavior of the memory mapper is altered.

Omitting the Characters and Using the Bits

In text generation systems, the microprocessor is responsible for placing ASCII codes into the desired locations in display memory. When the video display generator is ready to use a character, it fetches the ASCII code from display memory and then consults with a ROM chip to find out what pattern of dots it should send to the monitor to paint each of the seven lines which makes up the character (see Chapter 5, Figure 5.9).

For graphics, the microprocessor actually puts the dots directly into display memory. The patterns of ones and zeros are not codes at all, but are meant to actually control individual dots on the video screen. The Apple has a low resolution mode in which you can't control individual dots, but must work in clumps of 28 dots at a time. The reason Apple provides this low res mode will make more sense once graphics display memory is described. Very few Apple programmers will be interested in the low res system. Its only distinction is that each of its 40 by 48 equals 1,920 clumps or "pixels" can be drawn in any of 16 colors.

Representation of High Resolution Dots in Memory

In high resolution mode, each bit stored in memory controls one dot. When the video display generator fetches a byte of memory, it uses the stored bit pattern directly to drive the intensity signal to the CRT. The Apple high resolution system provides control of 280 dots on each of 192 scan lines, so there are 53,760 dots or pixels available for forming shapes.

The memory mapping of these dots in the motherboard RAM is sufficiently arcane and complex that it is not convenient for most programmers to attempt to plot points in memory (see Chapter 21, Figure 21.7). The Applesoft Interpreter ROM, however, contains several easy to use subroutines which let the machine language programmer get very good control. In Applesoft, control is absolutely effortless and, in fact, high res graphics in Applesoft BASIC is one of the outstanding features of this dialect of BASIC.

One very fancy and specialized way to load the dots into display memory is to plug in the Telidon Graphics System card from Norpak. This is a card with its own Motorola 6809 processor and 16K of machine language plotting routines stored in its own ROM. Just as ASCII code is used to represent letters and numbers, there is a code called NAPLPS which is made up of Picture Description Instructions (PDIs). This code system is used for telephone transmission of videotex pictures via modems using standard communications software. The Telidon card uses the 6809 to interpret incoming NAPLPS signals and to put bits in the proper locations in the Apple's high res display memory. It can also be used to create NAPLPS images in the Apple's hi-res display memory for later telephone transmission.

Double High Resolution in the II and II+

Although each byte of memory should control eight dots on the screen, Apple high res graphics only plots seven of the dots in each byte, reserving the eighth bit as a flag to keep track of which color to plot the points in. Within months of the first release of Apple's high res system, clever "hackers" had figured out how to reclaim the eighth dot for monochrome graphics.

This is not all that trivial because Apple uses the eighth dot to decide whether or not to slide the other seven dots to the right by half a position. These hackers realized that if a whole line was moved half a position, then it would be sitting in a completely different set of 280 positions than if it hadn't been slid over by half a dot. By clever control of the eighth bit, a programmer can get selective control over 280 normal positions and 280 half-shifted positions. This increases the Apple resolution to two times 280 equals 560 dots on a scan line in what is called "double hi-res mode."

Demo Program for Double Hi Res on the II/II+ or IIe/IIc

```
 90  HGR                          : REM ********************************
100  FOR Y= 0 TO 159              : REM ***  This section generates   ***
110  X% = 260 + Y/2               : REM ***  a series of points on a  ***
120  Y% = Y                       : REM ***     line, for the demo     ***
130  GOSUB 1000                   : REM ********************************
140  NEXT Y
150  HCOLOR = 7
170  FOR Y = 0 TO 159             : REM ********************************
180  X% = 100 + Y/4               : REM ***   This is just to draw a    ***
190  Y% = Y                       : REM ***  standard Hi Res line for   ***
200  HPLOT X%,Y%                  : REM ***  comparison on the screen   **
210  NEXT                         : REM ********************************
220  END

1000 REM ********** DOUBLE HI RES Point Plotter Routine ************************
1030 XX% = X%/4 : M% = X% - 4 * XX%
1040 IF M% = 0 THEN HCOLOR = 2    : REM ********************************
1050 IF M% = 1 THEN HCOLOR = 6    : REM ***  This section selects which  **
1060 IF M% = 2 THEN HCOLOR = 1    : REM ***  way to set the color phase  **
1070 IF M% = 3 THEN HCOLOR = 5    : REM ***     bit to smooth the plot    **
1080 HPLOT X%/2, Y%               : REM ********************************
1090 RETURN
```

Listing 7.1 Monochrome Double Hi-Res Plotter for the II/II+. This is a point plotter routine. Your program must generate the X, Y coordinates (X=0 to 559, Y=0 to 191). This method produces the same wide dots as standard hi-res, but it gives more precise control of point plotting locations.

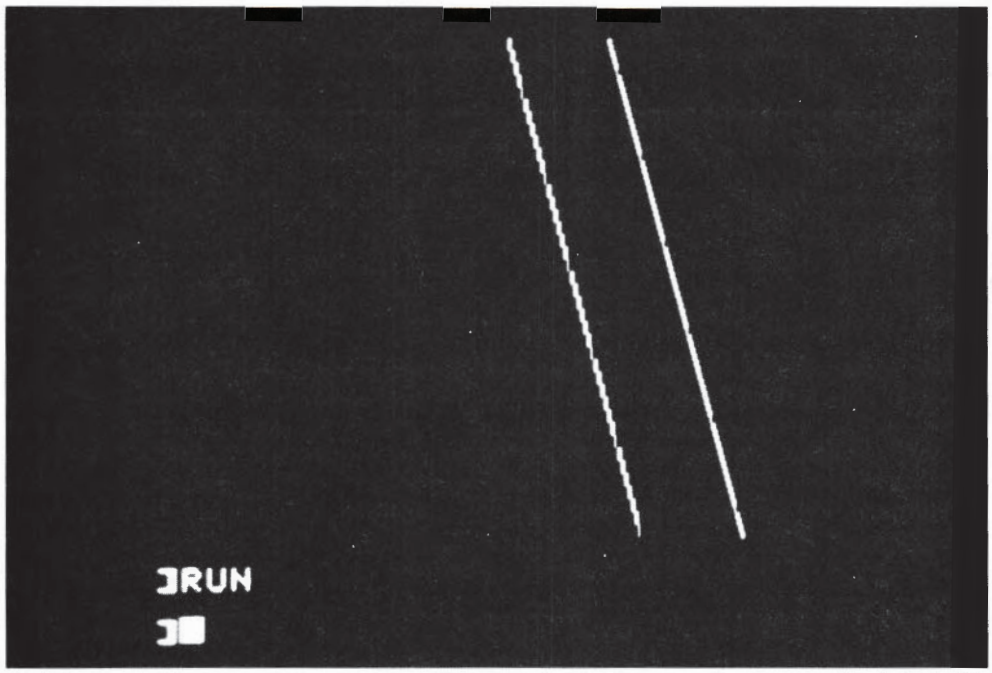


Fig. 7.1a (above) These lines demonstrate the II+ hi-res and double hi-res line. The dots are still twice as wide as in the //e and //c super hi-res, but the positioning is more precise than standard hi-res.

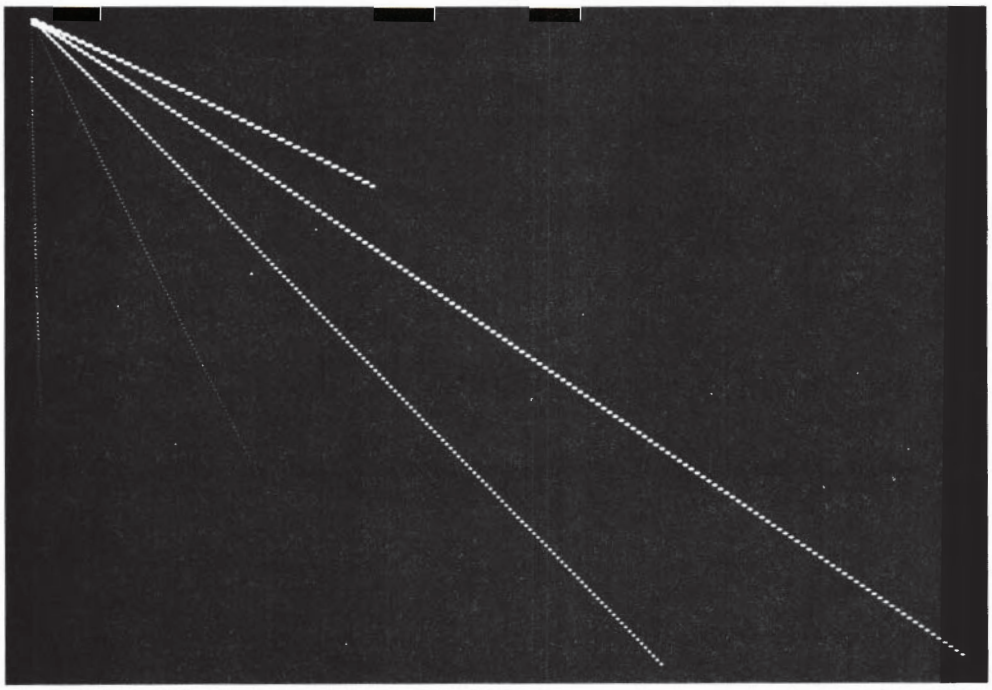


Fig. 7.1b (below) The super hi-res mode in the //e and //c has smaller dots and makes less jagged lines at low angles.

Listing 7.1 shows a very simple routine to add to a BASIC program which lets you plot in a field of 560 by 192 dots. It checks your horizontal coordinate, decides whether or not to shift it, and then accomplishes the shift by issuing a bogus color instruction to Applesoft before plotting. Figure 7.1 shows the high res and double high res lines generated by the program.

Super High Resolution for the //e and //c

If you own a //c, or have an Apple //e with a revision B or later board (see Appendix D), with an additional 64K of memory installed in the auxiliary slot, you can also get 560 dots per scan line in what is called "super high res mode." Various Apple documents have called this "double," "super," or "ultra" high res, but here we will refer to it as "super high res" mode.

This approach is completely different from the "double high res" mode just described. It uses the same bank switching system that the //e and //c use to display 80 columns of text instead of 40 (see Chapter 6) and has nothing to do with the eighth bit. Further, you get 16 true colors with full 560 by 192 dot resolution. Listing 7.2 is a listing of a program which can operate the super high res system. It accepts an x coordinate between 0 and 559, decides whether to put it in memory in the text card or on the motherboard, issues the appropriate bank switching commands, and then uses the appropriate Applesoft plotting command. The resulting super high res plot is shown in Figure 7.1b.

Demo Program for Super Hi Res on the IIe/IIc

```
 50  PRINT CHR$(4);"PR#3"
 60  PRINT CHR$(12)
 70  VTAB 22: HTAB 25
 80  PRINT "SUPER HI RES IGNORING COLOR"
 90  GOSUB 220                          : REM *** Initialize Super Hi Res ****

100  HCOLOR = 7                         : REM ********************************
110  FOR DX = 0 TO 10                   : REM *** This section generates  ****
120  CX = 0:Y = 0                       : REM ***  a series of points on  ****
130  FOR X = CX TO CX + DX - 1          : REM ***  on several lines and    ****
140  GOSUB 350                          : REM *** then calls the Super Hi ****
150  NEXT                               : REM ** Res point plotting routine. **
160   CX = CX + DX: Y = Y+ 1            : REM *** There is no equivalent of ***
170  IF X < 560 AND Y < 160 THEN 130    : REM ** the "HPLOT X1,Y1 TO X2,Y2" **
180  NEXT DX                            : REM *** command used in Hi Res ***
190  END                                : REM ********************************

200  REM******* SUPER HIGH RES Initialization and Screen Clear **************
210  REM
220  POKE 49154,0                       : REM ** RAMRD   Off **
230  POKE 49156,0                       : REM ** RAMWRT  Off **
240  POKE 49153,0                       : REM ** 80STORE  On **
250  POKE 49239,0                       : REM ** HIRES    On **
260  POKE 49160,0                       : REM ** ALTZP    Off **
270  POKE 49246,0                       : REM ** AN3      Off **
```

```
260  POKE 49232,0                    : REM ** GR            **
290  POKE 49165,0                    : REM ** 80COL     On **
300  HGR
310  POKE 49237,0                    : REM ** Aux.          **
320  CALL 62450                      : REM ** Interp. screen clear routine *

330  REM *********SUPER HI RES Point Plotter Routine**************************
340  REM
350  XX = INT(X/ 7): PG = XX/ 2 - INT (XX/ 2)  : REM*********************************
360  XX = INT(XX/ 2) + ((X/ 7) - XX)           : REM *** This section selects Aux. ****
370  XX = INT (XX * 7 + 0. 5)                  : Rem ***  or Main Memory for  ****
380  POKE 49237,0                              : REM ***  each point generated  ****
390  IF PG = 0. 5 THEN POKE 49236,0            : REM ***  by your program and  ****
400  IF XX > 279 THEN RETURN                   : REM ***  adjusts the plotting  ****
410  HPLOT XX,Y                                : REM ***       position         ****
420  POKE 49236,0                              : REM *********************************
430  RETURN
```

Listing 7.2 Super Hi-Res Plotter for the //e or //c (by Don Worth). There is a screen initialization section and a point plotter, but your program must generate the X,Y coordinates (X= 0 to 559, Y= 0 to 191). Dots are one-half the width of the standard hi-res and you can get 15 different colors. In order to run this program, DOS must be active and an Apple 80 column board must be present. See Solftalk, July 1983, page 121; Softalk, September 1983, page 83; and Apple Orchard, January 1984, page 26 for more elaborate routines and information.

Memory Mapping for Graphics

In text mode, there are just 24 by 40 equals 960 character positions on the screen. Each of the 960 characters is represented by a one byte ASCII code so the text display requires just about 1K bytes of memory. When a character is actually painted on the screen, it is made up in a field of eight scan lines of seven dots each (see Chapter 5, Figure 5.9), therefore each byte of text calls up a fixed pattern of eight by seven equals 56 dots. To paint a character on the screen, each ASCII code is called up eight times, once for each line of dots it requires to fill the matrix.

Lo-Res Display Memory

In lo-res graphics mode, the bit patterns are placed into the same 960 locations in memory that are used for ASCII codes in text mode. This practice dates back to the "early days" of Apple when many Apple owners had a grand total of only 4K or 8K of RAM memory installed (RAM was very expensive in those days).

To get 1,920 pixels from these 960 bytes, the lo-res system treats the first four bits of every byte differently than it treats the second four bits in the byte. The appropriate terms for these two halves of a byte are the low order "nybble" and the high order "nybble." The memory mapper fetches bytes from the same locations in display memory as it uses for text, but it does not just send the bit patterns directly to the intensity signal. During the first four fetches, it uses just the low order nybble, and during the second four fetches it uses only the high order nybble. Each of the 40 character positions on a line are thereby divided into a top pixel with four scan lines, and a bottom pixel with the next four scan lines.

Although the information for a low res pixel is contained in just four bits, the video display generator needs to do one more trick to use these bits to get any one of 16 colors. The reason for the maneuver is rooted in the deep technology of color television, but let's just state this for the record. The Apple switches into high speed by using the 14 MHz clock to send the dots out to the television set, and it "circulates" the four bits so that the whole nybble gets sent out four times in a row in a steady burst of 16 dots. The arrangement of the dot patterns is just so that the TV set typically thinks it is getting eight dots at seven MHz, not realizing that there are really 16 dots in the stream appearing at 14 MHz. Some of this makes more sense once the color scheme is laid out.

High Resolution Display Memory

The task for the memory mapper is really much more straightforward in high res. For each scan line it draws on the screen, it uses a different set of memory addresses. A screen scan line is still treated as if it has just 40 positions, but there are 192 such scan lines to be dealt with instead of just 24 rows as in text. A full screenful of high res data requires 40 by 192 equals 7,680 bytes of data; nearly 8K of display memory.

To draw a high res image, a byte is called up from display memory for each of the 40 positions on each scan line. The bits in that byte are used directly to turn on and off the intensity signal to the CRT. Since the eighth bit is reserved for color information, only seven bits are sent from each. A review of the addressing for one of the high res display memory ranges appears in Chapter 21, Figure 21.7, and there is a discussion of memory use for multiple display pages in Chapter 26.

One resemblance to the text system is worth noting. Recall that in 40 column mode, a seven MHz clock is used to shift out seven dots for each of the 40 characters. Similarly, the dots in high res graphics mode are shifted out at a rate of seven MHz.

This is the basis of the super high res mode provided by the //c and the //e revision B board. Much as in //e and //c 80 column mode, display memory is split between an 8K range on the motherboard and an 8K range on the extended text card or //c auxiliary RAM. When a display memory address is called by the video display generator, a byte from the auxiliary 64K card and a byte from main memory are simultaneously loaded into respective 74LS 374 chips. The two bytes are called up in two different halves of each microsecond by way of the video data bus, and the dots are clocked out to the CRT at 14 MHz.

This also means that super high res dots are smaller than the dots produced by normal or double high res. Because the clock rate is twice as fast (14 MHz as opposed to seven MHz), the electron beam is kicked up for only half as long for each dot.

Color

When Wozniak set out to design the Apple II's video display generator, he wasn't satisfied to just produce text, and he wasn't satisfied with high resolution graphics. The system had to do color graphics, and it should be able to operate a color television set directly. These are all demanding engineering tasks for a single integrated digital system, but the last was the most difficult. Wozniak's solution to the problem was the true "piece de resistance" in the design of the Apple. However, some may think he was a bit too ambitious.

To make the whole thing happen required a substantial number of design compromises elsewhere in the system. These compromises provide an endless source of challenges for peripheral designers who know about them and an endless source of puzzlement for some peripheral designers who don't. In any case, the whole thing flies (so to speak), and the overall verdict has been overwhelmingly positive. The Apple II video system is an electrical engineer's work of art.

Nonetheless, there are a variety of ways in which the Apple's color graphics can be enhanced by a number of different categories of plug-in cards. The following discussion will attempt to describe the Apple's color video for its own sake and to make clear what task each of these categories of peripherals is intended to accomplish.

Broadcast Color Conventions

The inside surface of a color monitor or television screen is quite different from a monochrome CRT. Dot locations on the color screen are real discrete things, and each dot is actually made up of three small patches of phosphor; one red, one green, and one blue. The electron beam itself is actually three separate but very closely aligned beams. When the triple beam points at a dot, each of the three beams points at just one of the patches of phosphor. No matter which dot the triple beam is pointing at, one beam always zeros in on the red dot, one on the green, and one on the blue. Therefore, although the three beams put out otherwise identical streams of electrons, one beam always causes red to appear on the screen, one beam always causes green, and the third always causes blue. For this reason, one speaks of a red beam, a green beam and a blue beam.

To transmit a color image you don't have to send the colors themselves, you just have to send information telling the television set what combination of the three beams it should turn on as it gets to each dot in its raster scan. You can, of course, generate any color in the rainbow with a proper mix of these three colors, and the dots are so tiny that the eye perceives a single hue rather than a pattern of red, green and blue dots.

It might have been simplest to just broadcast three separate signals, all closely synchronized and each controlling a different beam, but the dictates of assembling a reasonable radio frequency signal ruled out this option. Instead, the controlling information is sent as a single coded stream of information and then decoded by the television receiver at high speed as the beam sweeps across the screen.

This sort of code grows out of the days when electronics was all about resonant circuits and smoothly oscillating waves. The coding is done by fiddling with waveforms. The most easily grasped analogy is two runners of similar build jogging along side by side. Their legs are the same length, their running style is identical, they're moving at the same speed. You can imagine their perfectly synchronized footfalls coming down exactly at the same instant time after time. It could be said that they are moving their legs at the same frequency.

Now consider what would happen if you had the same two runners and all else was identical except that this time the two were a little out of step with each other. One runner's left foot is just starting to touch the ground at the same time that the other runner's left foot is pushing off. This relationship would be repeated identically for every step. In this case, you would say that they are running at the same frequency but they are a little "out of phase." This is how the coding is done for color broadcast.

The frequency used for color broadcasts is absolutely, strictly, exactly, and incontravertibly 3.579545 MHz. Why not a nice round number like 3.5 MHz or, better yet, 4 MHz? Well this number was chosen by a committee sometime in the 1950s. But in their defense it must be said that the frequency they chose was strictly dictated by some mathematical consequences of other nice round numbers used in broadcasting which had been chosen by others before them.

In order for any system to operate the decoding system in a color television set, it must be able to generate a signal at exactly 3.579545 MHz, no ifs, ands or buts. The coding system is based on detecting an incoming signal at that frequency and measuring how far out of phase it is with a preestablished standard. If it's a little out of phase call it a yellow, a bit more out of phase call it magenta, very far out of phase call it green, and so on. The television receiver does the phase measurement and decides what combination of the three beams to turn on at each dot. The broadcaster can cause the phase difference to change for each dot, and thereby gets control of the hue of each dot on the screen.

Apple Color Video

Since Wozniak knew he would need a signal at about 14 MHz and he knew he needed one signal at exactly 3.579545 MHz, he chose to make the master crystal run at $4 \times 3.579545 = 14.31818$. That is the source of this fundamental Apple number. Nearly all other timing signals in the Apple are synchronized with, and derived from, the 14 MHz signal, so everything in the Apple steps to a beat determined by the NTSC color committee many decades ago.

Now that you understand the importance of the 3.5 MHz signal, let's look at how the Apple actually sends the color code to the TV set. If a byte in high res display memory had the bit pattern "10101010," think about what would happen when it is sent to the monitor. High res bits are clocked out at a rate of seven MHz, which means an endless stream of this bit pattern would send out seven million bits every second.

But recall that the ones turn the beam on and the zeros turn it off. From the point of the TV set, you might consider the ones and zeros as peaks and valleys in a steady signal. In these terms, you could say that 3.5 million peaks (ones) would arrive each second. Yes, this is a wave with a frequency of 3.5 MHz (3.579545 MHz to be exact). If the byte was "01010101" instead (notice a zero on the left, the eighth bit, instead of a one), the frequency would be the same but the phase would be different. And you no doubt remember that that is how color is encoded.

Actually, only seven bits get sent so there are a few more tricks to get it all working right, but you should have the basic idea. A full explanation is available in Gayler's *Apple II Circuit Description* or Jim Sather's *Understanding the Apple II*.

Now what about the eighth bit? This is the one that gets fiddled with to get double high res graphics in an Apple II or II+. In monochrome, this bit causes a shift in position which can be used to double the resolution to 560 dots. But in color video, it's intended use is to provide a finer increment of phase shifting to get more colors. When this bit is set properly, the outgoing pulses are delayed for one 14 MHz pulse. This gives finer phase control for color and it permits finer position control for monochrome.

The Timing Compromises

This section is for the diehards who have been multiplying frequencies and counting scan lines to see how it all adds up. It is not an explanation, just a guide to the math.

If the Apple used the standard horizontal sweep rate of 15.734 kHz, then each sweep would take 63.5 microseconds. Most 6502 clock cycles in the Apple take 978 nanoseconds, so 63.556/ .978 equals almost exactly 65. This means that there are 65 potential character positions in every scan line. Each character gets seven dots at seven MHz so each dot takes 139.68 nanoseconds, seven dots take 0.97777 microseconds, and a full line of 65 by 7 dots takes 63.5555 microseconds. Each cycle at 3.579545 MHz takes 279.37 nanoseconds, so you can fit in exactly 227.5 cycles of the color reference clock.

The one problematic number from the above paragraph is the 227.5. This means that the color reference clock would be halfway through a cycle when the next line started—so its phase would be wrong. This trouble occurs in Apples because the character cell time from the 6502 runs off the same clock as the video. Wozniak's solution was to essentially shut the Apple off for two counts of the 14 MHz clock at the end of every horizontal scan line. During this time, the 3.5M signal could seem to completely finish its 227th cycle, and be ready to start the next line fully in phase.

The consequence of this is that every 65th clock cycle of the 6502 is a little longer than the rest. Further, each horizontal scan takes a little longer than it would in a normal video situation, and the horizontal scan rate drops from 15.734 kHz to 15.7 kHz.

Video Tape Recorders and Broadcast

For those who wisely skipped the preceeding bout of numerology, a good summary is that the Apple sends out its horizontal synchronization signals at a rate of 15,700 per second. This is a little less than the standard broadcast rate (15,734 kHz) agreed upon by the National Television Standards Committee (NTSC). Wozniak chose this non-standard rate as a compromise in order to make it possible to accomplish several other video timing tricks described above.

Further, because the Apple uses a non-interlaced display, (see Chapter 5) there are only 262 scan lines of output per field instead of the 262.5 in one half of an NTSC interlaced field. There are actually about 10 or 15 arcane aspects of video timing which are slightly non-standard in the Apple, and interested engineers should consult *The Apple II Circuit Description* by Winston Gayler (Sams and Co., Indianapolis) for a full accounting.

At the time, these slight differences seemed like no special problem because it was so close to NTSC standards that any monitor or television set would not notice the difference. However, modern Video Cassette Recorders (VCRs) are much more finicky and will not tolerate Apple's non-standard video signal.

Two companies, Adwar Video Corp. and Video Associates Labs, sell equipment which can help compensate. Adwar sells a device called the Proc Mod for about $100 which gives substantial compatibility with VCRs by fiddling around a bit with the synch signals. However, advertisers or TV newscasters who want to adapt an Apple to actually generate broadcast standard images must buy Adwar's more elaborate ARS 170 card which costs over $1000. This board actually captures two successive non-interlaced frames in its own memory and then recreates the video signal in full NTSC standard output, 15,734 kHz, 525 interlaced scan lines, and a 59.94 Hz vertical synch.

The full scale system from Adwar permits a video enthusiast to create labels and graphics with the Apple and overlay them on top of regular TV images on a VCR. With this special effects keyer, the entire system costs about $2000. Video Associates Labs makes a very similar system which sells for $2500.

This substantial cost has to be balanced against the $15 or $20,000 cost of a purpose-built system. Systems of this type have been purchased by TV stations, corporations preparing videotaped sales presentations, and schools wishing to economize in training students in both video and microcomputer technology.

Another option is to replace the Apple graphics system entirely by using a Number Nine Graphics generator board which can be made to produce true NTSC output as just one of a variety of features (see below). This approach does not permit you to use Apple graphics software you may already own, but there is a substantial amount of software available specifically for the Number Nine board.

RGB Boards and High Resolution Computer Graphics

Fortunately, most folks interested in good resolution color graphics don't need to use VCRs, video discs or broadcast equipment and can choose a very different route towards simplifying and improving the Apple video system. Most of the complexity of Apple color video is due to the desire to make the color signals acceptable to standard color television sets (after passage through an RF modulator). The Apple goes through a complex process of encoding the color information as phase variations at 3.5 MHz, and then the television set or color monitor has to turn around and decode the signal to see which electron beam should be on and when it should be on.

Since computer graphics need not accommodate the idiosyncrasies of broadcast video, it is possible to take a vastly simpler and more precise approach. The computer sends out three individual signal lines to directly control the three electron beams. There is no coding or decoding. There are several companies which make peripheral boards or //c expansion modules which decode the Apple's color signal and send it out as three separate signals, one for the red beam, one for the green beam, and one for the blue beam. These cards are, not surprisingly, called RGB boards.

Because there is less chance for ambiguity, color video based on RGB output signals tends to produce crisper images. Most of the color monitors on the market require RGB output, so an Apple owner often decides to buy an RGB board in order to use some specific monitor. However, //e and //c owners who wish to use the 16 color super high res feature will find that only RGB output is adequate.

Apple II and II+ owners can select from among RGB boards from Advanced Logic Systems, Amdek, Electrohome, Microtek, M & R, and from Telemax, but these boards differ in several respects. In addition to the red, green and blue signals, (and the horizontal and vertical synch) an RGB monitor can use an intensity signal. With three color inputs you can generate eight different colors, but an intensity signal can be used to get finer variation of the hues. Most of the Apple RGB boards are for monitors which accept a digital intensity signal; it is either on or off. Such digital intensity signal RGB boards can control 16 colors.

Digital RGB Boards

The RGB boards from ALS, Amdek, M & R, and the Telemax VCB A2 all provide digital intensity control. Among these, the Amdek DVM II board stands out because of the way it does the decoding of the Apple output (see Figure 7.2). Most RGB boards do the decoding by comparing the Apple's video output to the Apple's regular 3.5 MHz color reference signal.

There are two problems with this approach. First, that signal is only available to peripheral boards plugged into peripheral slot 7 and is not available on any of the regular peripheral slots in the //e. So most of the RGB boards can only be used in slot 7 of an Apple II or II+. The second problem is that using the Apple's color reference signal apparently leads to some of the same occasional ambiguities that the RGB system is meant to avoid.

The Amdek DVM II RGB board uses the signal directly from the Apple's 14 MHz crystal. This means you have to do some fiddling on the motherboard, but the accuracy is the best possible, and you can put it in any slot. It is also possible to take the output from your 80 column text card and send it into a port on the DVM II. The board will respond to commands from software to stop displaying color graphics and start displaying clean, sharp 80 column text in any of 16 colors or clean white on black with a good quality color monitor.

Analog RGB Boards

Another feature provided by some RGB boards is fine control over the intensity signal. The Supercolor Board from Electrohome and the Rainbow 256 from Microtek can both produce 256 different colors on the screen of an RGB monitor which can accept an "analog" intensity signal. This is very nice, but you have to use special software to take advantage of the colors because the Apple video system itself has no way of specifying more than six colors in high res mode or 16 colors in low res mode. Both of these boards must be installed in slot 7 of a II or II+.

RGB for the //e and //c

If you want RGB output in a revision A //e, you're probably going to have to put your RGB board in the auxiliary slot, because none of the II/II+ RGB boards will work in the //e. This is because Apple had removed the 3.5 MHz color reference signal from peripheral slot 7 in order to use that pin for a new signal.

Fortunately, Apple has provided a rich variety of timing signals on the pins of the auxiliary slot, and Amdek, Telemax, and Taxan have come to market with RGB boards for the //e auxiliary slot. All of these boards permit the display of 80 column text in any of 16 colors or white on your RGB monitor. The Amdek DVM 80e, the Telemax A2e, and the Taxan 410-80, all provide RGB and //e 80 column support. Taxan has also released the "410-64," which includes both an extra 64K of auxiliary RAM and a digital-intensity RGB output.

All of the important signals for generating RGB color output on the //e auxiliary slot have been led out to the external video expansion connector on the //c (see Figure 7.2b and Appendix A). Therefore you still must attach an RGB module such as the Taxan 410-15 to translate the connector's timing signals into true RGB output.

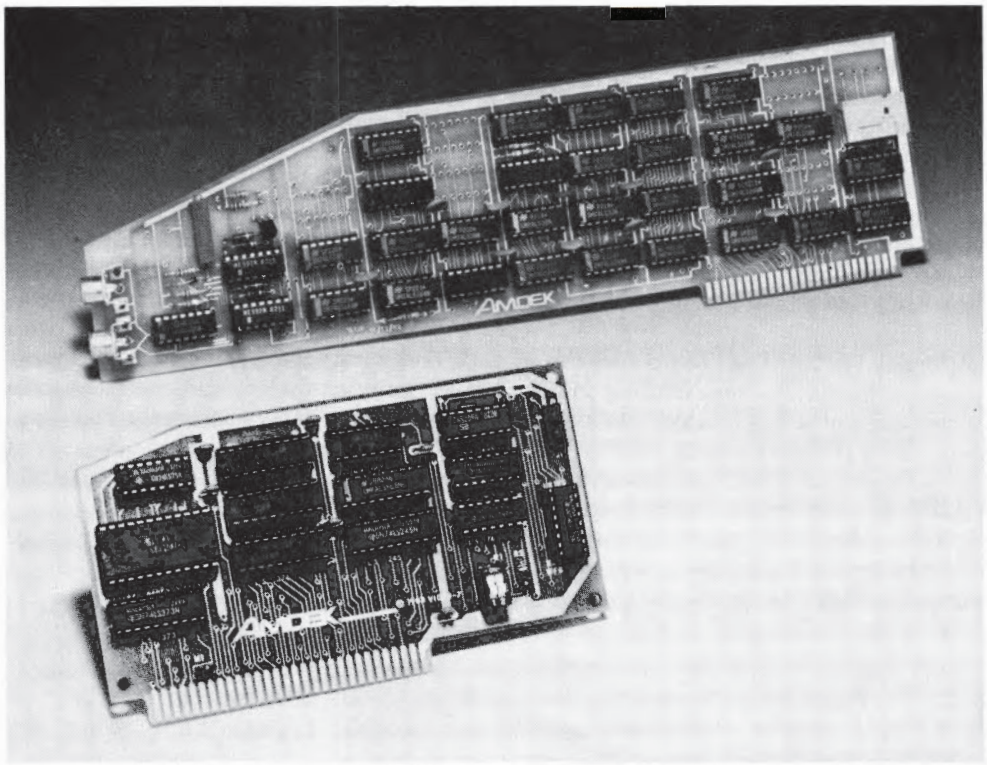


Fig. 7.2a The Amdek DVM II (top) and DVM 80e (bottom). Both boards provide RGB output, but the DVM 80e provides simple switching among //e video modes.

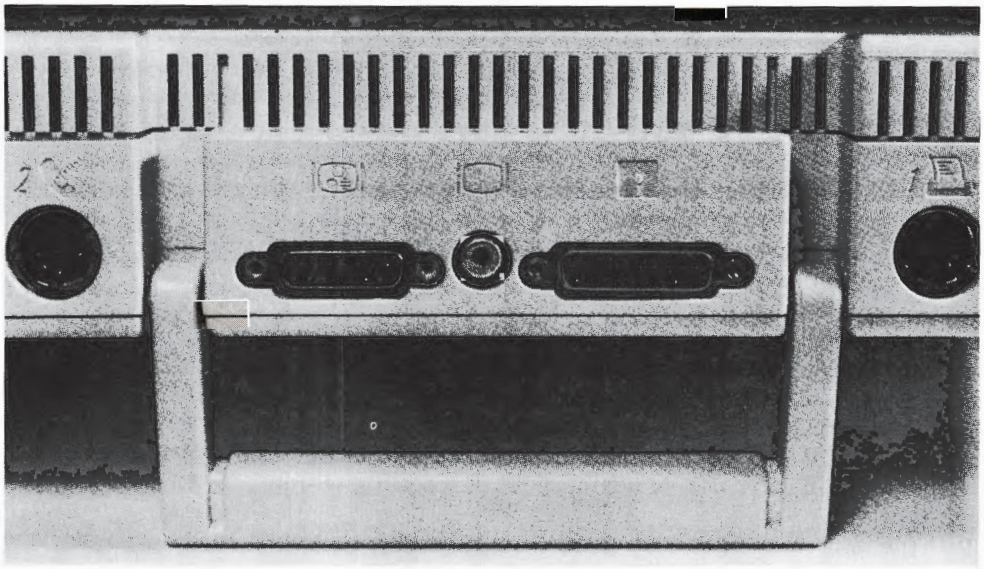


Fig. 7.2b The //c video expansion connector provides some of the same video signals as the //e auxilliary slot. Therefore, you need an expansion module to actually get RGB output, just as //e owners need an RGB card.

Color Graphics Generators and Ultra High Res

All of the color and graphics peripherals discussed so far have been used to extend the capabilities of the Apple's built-in system for generating high res color graphics. However, it is possible to install a peripheral board which produces graphics all by itself, independent of the Apple high res video display generator. This is conceptually very much like the idea of installing an 80 column card to be used instead of the Apple's own 40 column text generator, except that you replace and upgrade high res graphics instead. Just as the 80 column text cards are based on single chip CRT controllers, these graphics boards are based on single chip graphics display controllers.

These graphics boards make improvements in several problem areas. The first is that they can produce standard NTSC video signals at a much lower cost than the corrective boards from Adwar and Video Associates Labs, and can therefore be used with VCR and video broadcast equipment. One of the boards offers options for any or all of true NTSC composite video signals, RGB with digital intensity or RGB with analog intensity.

Second, they can add substantial new capabilities for the graphics programmer, including the ability to produce high speed animation. You see, in order to plot images with Apple high res graphics, the programmer must work with a few simple geometrical commands. Once a shape is formed, every movement of the shape must be treated as a problem of complete geometrical recalculation of each point. Although much of this work is done by the Apple's own machine language subroutines, it still means substantial work for the programmer and substantial processing time for the 6502. This is why Apple graphics is tied to fairly slow, coarse movements on the screen.

The emphasis in graphics chips is to let the programmer refer to shapes, regions and directions of movement rather than to the dots themselves. The programmer states what he or she would like to see happen at a fairly high level, and the chip acts on the commands as simple machine language commands in its own graphics oriented machine language. In the 6502, machine language commands refer to memory addresses and registers; but in a graphics controller chip, the machine language commands refer to shapes and motions directly. This all results in an enormous improvement in speed of drawing and of animation of shapes. The Number Nine Graphics System board can cause complex multi-colored objects to appear to spin rapidly on the screen.

Third, and finally, a graphics board can provide additional display memory and the capability to deal with much higher numbers of dots. The Number Nine board provides 128K of RAM, devoted exclusively to its own graphics display needs. Of course this means that Applesoft programmers will not have to reserve space in the main memory for the Apple graphics display, but the true dividend is in the remarkable increase in resolution. Apple high res gets you about 54,000 dots in monochrome or about 27,000 dots with six colors. The Number Nine board can get you over a million dots in monochrome or 260,000 dots with 16 colors. That's right folks, an order of magnitude (10 times) increase in resolution for color, and 20 times the resolution in monochrome (this sort of stuff requires a very good monitor though).

It should be noted that as soon as you break free of the Apple's own graphics, you also lose the ability to use the Apple's simple graphics commands. These graphics boards are only as good as the available software and documentation. Fortunately, the Number Nine board has generated so much excitement among graphics programmers that the software situation is already quite excellent.

Programmers are provided with a set of new high level commands to use from within Applesoft BASIC, and another company offers a set of commands to be used from Pascal. Integrated packages are available for "paintbrush" work with graphics tablets, for commercial artists who need to manipulate a variety of type fonts, for video camera digitizing, and for hard copy printout on high res color graphics printers.

One of the most exciting implications of the arrival of the NNGS board is the advent of high quality Computer Aided Design (CAD) and architectural drafting on a personal computer system at one tenth the cost of current CAD systems. Standard CAD software is currently being rewritten to run on the Apple.

The Graphics Chips

The graphics boards for the Apple are sharply divided by their use of two different generations of graphics chips. The less expensive boards use the older TMS 9918, but the truly spectacular Number Nine board uses the new NEC 7220 graphics display controller.

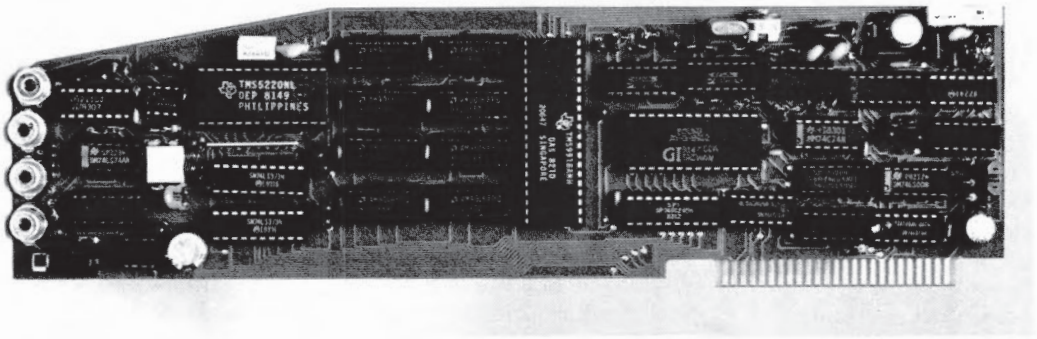


Fig. 7.3 Super Sprite from Synetix. A TMS 9918 generates the video sprites, an AY-3-8912 produces sound effects, and a TMS 5220 generates speech.

Sprites and the TMS 9918

The two boards which use the TMS 9918 are sold by Synetix (Super Sprite; $395) and by Third Millenium Engineering (The Arcade Board; $295). Both of these boards include 16K of display memory and also have facilities for generating special effects sounds.

The TMS 9918 does provide for a modest increase in resolution over standard Apple color high res (260 by 192 with 16 colors versus 140 by 192 with eight colors), but it's real advantage is that it moves much of the work of replotting points into the hardware realm of the chip instead of depending on the 6502 running software. As a result, you can use the TMS 9918 to create shapes and then cause them to move very rapidly across the screen. Motions can take place in 32 different imaginary planes—as if each of several objects was moving within its own transparent overlay sheet.

The Super Sprite (see Figure 7.3) permits you to overlay the moving images (sprites) on top of an image generated by standard Apple high res graphics. The Super Sprite achieves partial compatibility with video cassette recorders (VCRs) by stripping out the Apple's non-standard

horizontal synch pulses and replacing them with NTSC standard synchs at about 15.73 kHz. However, this is not full standard NTSC output. You can use a VCR, but you cannot mix Super Sprite output with standard NTSC video or use it for broadcast.

The Arcade Board does not use the Apple's high res output at all, but it does permit you to link two arcade boards together, and it does produce full NTSC standard video output.

Ultra High Resolution and the NEC 7220

The Number Nine Graphics System board (see Figure 7.4) uses an extremely powerful new chip called the NEC 7220 Graphics Display Controller (GDC). Until 1983, a graphics output of this quality could only be produced by a $25 to $30,000 committed system or an enormously expensive mainframe computer. The NNGS board sells for $945 in its simplest configuration (512 by 512 with 16 colors and 724 by 724 monochrome), and a $200 upgrade provides programmable analog RGB output which permits the board to control 4,096 colors at each dot. An alternative upgrade ($145) provides full NTSC standard video output for complete compatibility with all VCR, video disk, and broadcast requirements. There is a light pen interface, and Number Nine also provides a connector for an optional external power supply.

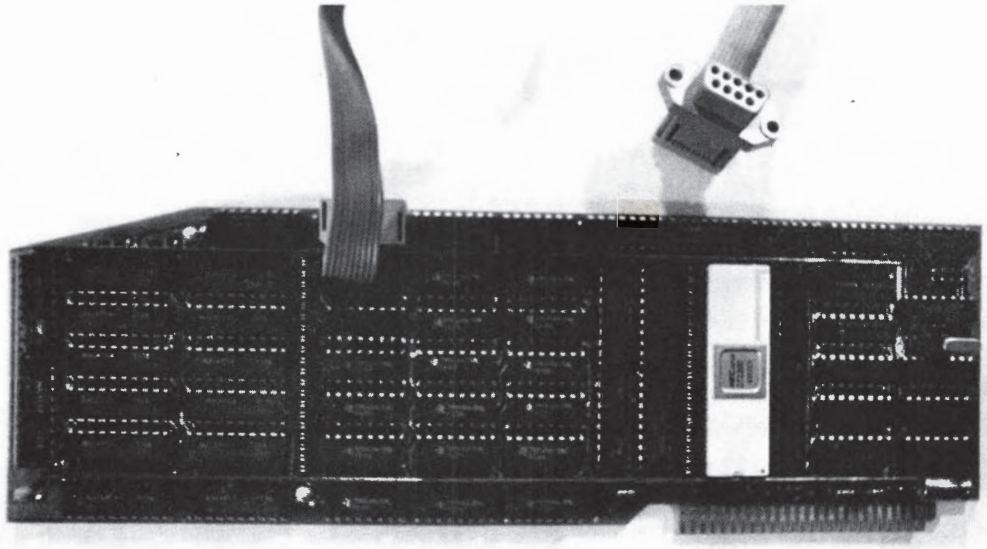


Fig. 7.4 Number Nine Graphics System for ultra hi-res video. The board has a NEC 7220 graphics display controller and 128K of RAM for display memory.

Objects form, fill, and move with remarkable speed in the native board, but the upgrade permits you to form a complete image in one of the four color planes, keep it invisible until it is drawn, and then suddenly reveal the complex shape in a matter of milliseconds. This is just one of those things that has to be seen to be believed.

A company with extremely demanding graphics needs can purchase up to four of these boards for use in coordination in a single Apple. By choosing the high speed version of the board, and adding the analog intensity RGB output module, you can produce 1024 by 1024 pixel resolution with each dot being displayed in any one of 4,096 colors.

The NEC 7220 machine language includes commands for a variety of geometric shapes and performs high speed "vector math," which is how it can move hundreds of thousands of dots at a time at high speed. Because there are machine commands for zooming and panning, it is possible to use some of the typesetting fonts to create words written in small letters which rapidly grow and move across the screen. This can all be filmed with a movie camera pointing at the screen or sent directly to a VCR as video.

A similar but slightly more elaborate board from Micrographic Images has an interface with a high speed video digitizer for rapid capture of images from camera to the NEC 7220 display memory.

If you need this level of resolution but you don't need all the bells and whistles of the NEC 7220's vector math and shape commands, you might consider the Rana 8086/2 (see Chapter 30). This is effectively a complete IBM PC compatible computer built to interface with the Apple bus. It differs from the PC in having two very high resolution modes, including a 600 by 400 dot monochrome mode.

Color Monitors

Unlike the situation with monochrome monitors, you must be very careful about which kind of color monitor you buy. It is not just a matter of which is "best," but a matter of the exact kind of use you hope to make of it. There are good reasons for buying any one of a low resolution, medium resolution, high resolution, or very high resolution color monitors.

Fig. 7.5 Color I low resolution monitor with composite video input. This is an appropriate monitor for standard Apple graphics, and includes a built-in speaker to amplify sound effects.

When Apple high res graphics is used to produce color images, it has an effective resolution of 140 pixels. This level of detail is easily accommodated by any low resolution color monitor such as the Amdek Color I ($380; see Figure 7.5), the Electrohome ECM 1302-1 (with composite adapter), or a standard color television set. All of these accept the single line "composite" output direct from the Apple and provide about 300 by 260 pixels. This is also adequate for output from graphics cards using the TMS 9918 chip. If you use this system, however, you cannot display 80 column text on the same monitor.

With an RGB board installed, you can use an RGB low resolution monitor such as the Amdek Color III ($479). Because of the improved control provided by the RGB output signals, this type of monitor can also be used for 80 column text if your 80 column card produces simple five by seven matrix characters. This applies particularly to owners of the Apple //e or //c. As explained earlier, //e owners may well choose to use a combined RGB and //e-80 column board. This system will work with a low resolution RGB monitor.

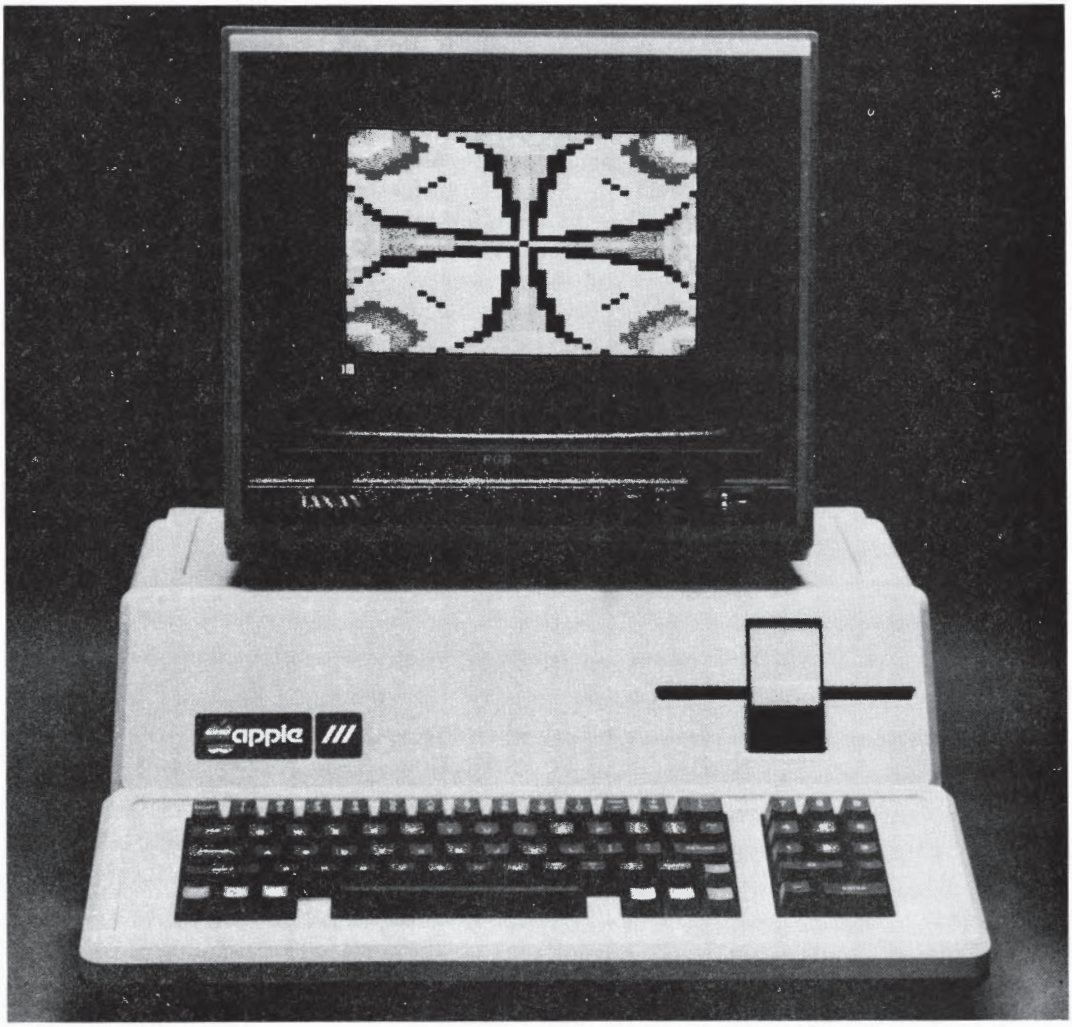


Fig. 7.6 Taxan RGB III high resolution monitor. This sort of monitor is for a special high resolution video generators or for extra clarity when 80 column text is displayed in color.

Medium resolution RGB monitors such as the Amdek Color II (560 by 240 pixels; $529) or Electrohome ECM 1302-2 have been designed primarily for the IBM PC market, but there are two kinds of uses for these monitors for Apple II owners. The Amdek DVM II board can be used to send 80 column display to an RGB monitor, but if the 80 column card uses a seven by nine character matrix, the dot pattern will not plot well on a low resolution monitor. The other use is for super high res graphics on //c's and revision B Apple //e's.

High Resolution RGB monitors such as the Taxan RGB III ($700; see Figure 7.6), the Quadchrome Monitor from Qaudram ($795), and the Amdek Color IV (with analog color range) are only appropriate if you plan to buy an NNGS graphics generator (or if you want to display 80 column output from the ALS Smarterm I via the Amdek DVM II card, but it's much cheaper just to buy a separate Amdek 300A monochrome monitor for ALS 80 column).

The simplest configuration of the NNGS graphics board was described as 512 by 512 with 16 colors. Actually, the exact proportions of horizontal and vertical must be changed a bit for use with one of these monitors. The standard configuration as shipped by Number Nine Computer Engineering is 620 by 420 and this is accommodated by all three of these monitors.

If you want to go beyond 620 by 420 in color, you're off into a different world as far as monitors are concerned. This puts you out of the consumer market, and prices jump sharply into the range of $2000 to $5000. You could contact Tektronix, but keep in mind that you're speaking of higher resolution than they provide in their 4100 series graphics editors. Ultra high resolution monitors are available from Hitachi and a few other sources, but it is probably best to discuss your application with the manufacturers of the NNGS system.

PART 2

The Interactive Apple

- **CHAPTER 8 The Human Interface: Keyboards, Pointers and Digitizers**
- **CHAPTER 9 Spatial Input**
- **CHAPTER 10 Sound and Music**
- **CHAPTER 11 Apple Speech and Hearing**

Chapter 8

The Human Interface: Keyboards, Pointers and Digitizers

The search for the ultimate input device is roaring along at full pace with mice, touch tablets, voice recognition systems, and light pens tossed into the stew to compete with the QWERTY keyboard. To add a bit more spice, computer designers, programmers and electrical engineers have recently begun making appeals to "right brain" versus "left brain" behavior in championing one input device or another. In fact, the abstracts of behavioral biology may have very little to tell us about how to design computers; however, since at least part of this writing team has spent the last four or five years at Harvard University doing research and teaching about the evolution and biology of human communication, this seemed to be as good a place as any to slip back into our own field of expertise and throw a few more brain terms into the debate.

You certainly can't do justice to the neurobiology of communication and perception in the space of three or four paragraphs, but a few points are worth making. Most of the attention in the design of human/computer interfaces goes towards what may be called "cortical" processes such as language. The traffic of human thought is carried out with the aid of symbols. In the left cerebral cortex of the brain, the learned words of a language such as English are used to represent ideas; in the right cerebral cortex, icons, images and patterns are more important. A person with damage to the right cerebral cortex may not be able to recognize whose handwriting they are looking at, while similar damage on the left side can block understanding of the written word altogether.

However, the elusive "open loop" in the human/computer interface may also involve other kinds of processing in the brain. There is a "limbic system" which pertains mostly to emotion and through which we attach importance to what we perceive in our environment. Even deeper within the brain is a "collicular" system which mediates spatial attention. A flashing cursor can catch our attention at a collicular level, causing a turn of the head or a rapid reaiming of the eye (which you don't usually perceive consciously) without the more ponderous involvement of the cortex. If the limbic system attaches importance to the flashing cursor, then the cortex may get involved.

Finally, at the level of muscular movement or "motor output," there is a distinction between unique or newly learned movements which must be stepped through, motion by motion, under direct conscious control of the cortex, as opposed to repetitive, well known movements which become encoded as nearly permanent patterns in a deep brain structure called the cerebellum.

A learned, cerebellar movement can be called into action by an effective command from any of the conscious thought of the cortex, the emotional response of the limbic system, or the instantaneous orienting response of the colliculus. These cerebellar movements can be rapid, smooth and precise, virtually independent of the complexity of the muscular or sensory acrobatics involved. (For electrical engineers, a useful metaphor would be to think of the cerebellum as a Programmed Logic Array (PLA) to the cortex's microprocessor.) The fastest responses of a highly skilled video arcade game freak (the kind who has been raised up from infancy with a joystick in hand) can reflect a "collicular/cerebellar" loop which never involves the slower associative processing of the cortex.

The attempt to develop "user friendly" windowing screens can be more explicitly approached as a problem in making the screen "collicular friendly." The colliculus has "hard wired" ways of making the eye scan surrounding space which involves a preference for tracing the edges of objects and for creating visual abbreviations for complex shapes. The brain perceives Lisa's windows as a small number of discrete objects; much of the work of selecting a single window to pay attention to is done at the level of the colliculus and limbic system before the cortical visual recognition process is fully engaged. The very different design of windows in, for example, Perfect Writer fails in this regard. The screen is perceived as a detailed array of complex shapes which must be picked through consciously by the cortex again and again.

Further, once a Lisa window is attended to, right cortex association of the icons can further relieve the burden of the left cortex which is trying to attend to words and numbers. This is in part why competing windowing software for the IBM PC, which does not account for all these fine points, often fails miserably, relative to Lisa, by making the computer seem more difficult to use and more taxing to attend to. The magic of Lisa is not simply due to crowding a large number of competing options into your field of view, rather it has to do with providing spontaneity of switching between tasks while retaining the sensation of a simple, uncluttered screen. Whether the designers of Lisa (and Xerox Star) arrived at their screen layout through an analytic approach to the literature of perceptual psychology or in response to intuitive aesthetics, many of the particulars of the graphics are not as trivial as they may seem.

Entering Words, Letters and Numbers

This brings us to the design of input devices. To get words and numbers into the computer your options include, voice, writing, typing, pencilled cards, and the primitive "front panel."

For voice, the Apple must digitize or acquire the sound pattern, undergo a pattern recognition process, and then pass along the selected string of ASCII characters. Writing on an input pad requires the computer to undertake a similar series of interpretive events, but although the word is already available in its component letters, computers have a much harder time with image recognition than with sound recognition.

The remaining options all shift much of the burden of decoding the word into letters on over to the human who is generally much better equipped than a microcomputer to carry out pattern recognition. Skilled typists can pound the keyboard as well as they can write and as fast as they can talk, so this distribution of labor has served the computer/human interface well for many years. Most modern keyboards do some further processing of their own to match keypresses with ASCII codes to feed to the computer, so everything proceeds quite smoothly.

Back in the days before we had inexpensive smart keyboards, and before IBM felt threatened by the microcomputer (i.e., circa 1980), most people who used computers did even more of the

work by pencilling in little boxes on computer cards or did daily battle with a grinding, whirring keypunch. You can, in fact, buy a punched/pencilled-card reader for the Apple from Mountain Computer (the Model 1100A), or from Chatsworth Data Corp. (OMR-500 and OMR-2000).

Twenty-five to 30 years earlier, punched cards were a big user friendly breakthrough which relieved computer operators from the necessity of sitting in front of a row of eight switches called a front panel and setting the digital pattern of each letter one at a time. You can still rig up your Apple to accept bytes in this fashion using some of the special digital ports mentioned in Chapter 15, but really now, let's not get carried away with nostalgia.

Entering Spatial Information

While voice input is still in an early growth stage (see Chapter 11), keyboard input is, for the most part, in a happy and mature state. Within the past year or so, however, one weakness in keyboard (and, for that matter, voice) input has gotten increasing attention. To wit, there are things we wish to communicate to the computer which are not directly equated with words. If you've ever found it bothersome to reach over to an arrow key to move the cursor during word processing, you might be crushed to learn that saying "left, left, left, up, up, up, left" into a voice system is infinitely more tedious and distracting.

This is the point at which computer designers have invoked the idea of left brain versus right brain. It is considered that the words and numbers on the screen have meaning in the left cortex of the brain, but that positions and directions exist in the "cognitive maps" and spatial notations of the right cortex. A "pointing" device (mouse, touch pad) lets you carry out your intentions for motion in the same terms in which you perceive the space on the screen.

The screen metaphor in Lisa takes this one step farther by representing menu options as pictorial icons in standard positions on screen windows. This is an attempt at a sort of "parallel processing" in which as many tasks as possible are allocated completely to what is assumed to be right brain processing so that they will not complicate the processing of words and numbers by the left brain.

Entering Affective Information

Still largely untapped are our emotional (limbic) and orientational (collicular) forms of expression. One of the problematic aspects of current voice input systems is that as the user becomes frustrated or nervous their tone of voice changes, and the computer becomes unable to recognize what they are saying, thus adding to the anxiety. It may, in fact, be this unresponsiveness to emotional tone that makes some people so uneasy with computers. The programmers at Electronic Arts claim to be quite interested in this problem. Further, it is interesting to note that the Alpha Syntauri SM 05 music keyboard (see Chapter 10) has velocity sensitive keys so the computer knows not only which key you struck, but how hard you hit it.

While there is no forseeable prospect of computers receiving our thoughts directly, one path for making a computer instantly aware of what a human is interested in is to track eye movements. The colliculus has a standard set of rules for directing scanning, but it does its work interactively with the limbic system which attaches importance to objects. All of this seems to imply that a computer can be made to be more responsive than most would like to contemplate. But, as we have seen, with enough RAM, a fast enough processor and just the right sensors, almost anything is possible (after all, this is 1984).

Fig. 8.1a (top) The Apple //e keyboard.

Fig. 8.1b (center) The Apple II keyboard.

Fig. 8.1c (bottom) The Apple //c keyboard.

Keyboards

The elements of keyboard design grow out of more than a century of mechanical constraints and cultural habits. However, in the past five years, the occupational health concerns of workers and the marketing concerns of manufacturers have combined to make this familiar array of push button switches into one of the most intensely studied objects humans have ever laid their hands on. Further, during the same five years, advances in electronics have greatly increased our expectations of the kinds of things a keyboard should be able to do.

The physical design of a keyboard usually takes the needs of a skilled touch typist as its starting point. You then throw in a few adjustments for the needs of less skilled typists who don't necessarily want to use a computer as if it were a typewriter, account for mechanical reliability, and make a few allowances for improved safety.

There are really two quite distinct areas of concern in the design of the electronics. The most fundamental problem is to mesh the peculiarities of the speed and regularity of human movement with the respective peculiarities of computer data acquisition. Beyond that, there is the issue of encoding letters as ASCII codes. While simple one to one matching has been the norm in the past, newer keyboards have their own microprocessors, RAM, ROM and audio systems, and perform services not anticipated in the age of the typewriter.

Ergonomics

If you ever get to compare the feel of the keyboard of an early Apple II to the one that was being installed in the last few II+'s coming off the assembly line in late 1982, you'll notice that a great deal had already happened before the arrival of the //e and //c. This is the impact of "ergonomics."

Aside from being a very trendy term, ergonomics, or "human factors design," is concerned with making you feel comfortable while you work. During the history of the II/II+, Apple Computer was both affected by and actively involved in efforts to improve keyboard comfort. The later keyboards reflect changes in the height and angle of the keytops in each of the rows of keys, a change from center-depressed to sculpted keytops, replacement of shiny plastic with a matte finish, as well as changes in how hard your fingers had to press and how far the keys travel.

Later versions of the //e extended this series of changes by using dark labels on the light matte background and adding small, raised, positioning dots on the d, k, and right arrow keys. The //c adds firmer tactile feedback, crisper audible feedback, and a rubber splash screen beneath the keys to catch smaller spills harmlessly (see Figure 8.1).

Mechanical Features

The most active research on keyboard comfort has been done by several German groups. One result is a set of Deutsche Industrie Norm (DIN) standards which are having such a substantial impact in the U.S. that they induced IBM to abandon its much loved Selectric style keyboard for a DIN style device for the IBM PC. You can get DIN style keyboards for the Apple II/II+ from Keytronics and from Multitech.

The purely mechanical standards cover the recommended height of the "home row" (ASDFGH, etc.) above the table top, how far you have to push a key before the switch throws ("pretravel"), how hard you have to push to get it there (a standard "activation force" for regular keys and a greater one for the space bar), and the maximum distance you can push a key before it hits bottom ("travel"). There are also suggestions on the shape and color of the key tops, and the angle of slope of the whole keyboard.

The DIN specifications recommend that the user be allowed to move and position the keyboard relative to any CRT screen. This is widely interpreted as a requirement for detachable keyboards, but it was really directed at the widely used CRT terminals in which the keyboard was built into the same unit as the screen. This forced the user to always stare at the screen from a fixed distance and at the same angle, causing eye strain. This problem does not necessarily apply to the Apple II since the monitor is not attached, thus the Apple II design meets the intention of the DIN specification. In fact, the great bulk of people who use the Apple II or the IBM PC work with the keyboard in a fixed position directly in front of the screen anyway, but this is not necessary nor is it always a good idea. The //c certainly meets the letter and the intent of the specification, and also fits handily in your lap while you're working.

The principal things you can't do with an Apple II, II+ or //e keyboard in this regard are to sit back with your feet up and the keyboard in your lap, much as some folks like to do when writing in a notebook, nor can you move the Apple cabinet and drives off of your desk to reserve space only for the keyboard. For the large majority who prefer to work at a desk, the Apple II design works very well. However, for those who want a detachable keyboard so they can put their feet up or make more space on the desk, there are a number of them available.

Feedback

Another important concern has to do with "feedback" which helps you to be certain you have activated the key. This was never a problem for typewriters which made an unmistakable clatter for each keystroke. Among computer users, this has turned out to be a matter of substantial differences in personal taste. A few companies make completely silent keyboards (i.e., Keytronic) where the only feedback is the steadily increasing resistance as the key descends ("tactile feedback").

Most Selectric or Apple style keyboards rely on an audible confirmatory snap either as the key strikes bottom or as it pops back up after release. A third option which is built-in on some keyboards or can be generated from software in the Apple is a click sound (10 to 40 milliseconds at 500 to 1500 Hz in some Keytronic keyboards) each time the computer receives a character.

The PCPI Appli-Card Z-80B coprocessor (see Chapter 29) lets each user select and set a preferred volume for the click or no click at all. You make your choice and then it is patched into the CP/M operating system disk and activated every time you boot up. The click is generated from the Apple's speaker regardless of the kind of keyboard you are using. This has the provocative and interesting consequence that each user can set up their own version of the PCPI CP/M operating system according to personal aesthetics.

Secondary Key Blocks

The major point at which skilled touch typists part company with the remainder of computer users is in their attitude towards additional blocks of keys placed out of reach of the home typing position. Typists using WordStar will happily type four and five key control sequences to issue commands as long as they don't have to do a hunt and peck cycle which involves looking at the keyboard, reaching, and then returning to the home position. More and more

computer users, however, are coming to prefer special "function" keys which help avoid the tedium of learning and typing command sequences. Function keys are often arranged as a separate group along the top of the keyboard or on either side of the primary key area.

In addition to command function keys, many users like to have a separate cursor movement pad to tap at without concern for accidentally hitting other keys. A "compass" shaped cursor pad has the added advantage of purely spatial correlation with the screen. Finally, there is substantial interest in numeric key areas. Many typists who otherwise are primary key fanatics do not like entering large amounts of numeric data with the normal number keys.

The most popular arrangement for number keys is called a calculator style arrangement (7, 8, 9 on the top row) though some like a telephone style arrangement (1, 2, 3 on the top). More recently, the demand has been for "VisiCalc pads" which should include not only numbers and arithmetic operators, but also parentheses, label, direction, and command keys in a cluster for operation with one hand. Variations on these themes are available from several companies for addition to the II/II+ or //e (see below).

Keyboard Electronics to Interface Humans with Computers

The electronic system that manages the keyboard must mediate among the way humans type and the way mechanical keyboards work and the way the Apple reads in data. The whole process involves detecting a keypress and translating it into a digital code, isolating the computer from the slow and irregular pace of the human, and responding to requests for rapid repeat. The translation step involves correctly detecting a keypress, identifying which key it is that's down, selecting the appropriate matching output code, and informing the 6502 that a new code is available.

Things get a little trickier a few millionths of a second later, because the human finger that started the sequence is most likely still engaged in the process of pushing the key down. The entire keystroke may take over half a second (which is 500,000 cycles of the 6502's clock). To avoid any ambiguities, the keyboard electronics must detect both the beginning and the end of a single keystroke and treat the whole period of time that the key is down as if it were a single rapid event lasting less than a thousandth of a second.

If the human decides to activate the repeat or autorepeat (a //e and //c) feature, another problem arises. Operating at a wide open, uninhibited rate, the keyboard could send a thousand copies of the character per second. This would rapidly fill the screen and overwhelm most running programs in the computer. There is therefore a special electronic subsystem to handle autorepeat and slow it down sufficiently to properly emulate the approximate typing speed of a human.

The AY-3600 Keyboard Decoder

Each key on the keyboard sits above a position on an X,Y grid made up of electrically conducting X lines and Y lines. When a key is pushed, it throws a switch connecting one of the X lines to one of the Y lines (see Figure 8.2). If the keyboard electronics can tell which X line is connected to which Y line, then it will know which key has been pressed.

At the heart of the detection and translation system are two counting registers in a chip called the AY-3600 Keyboard Decoder. This chip is used in the //c, the //e and in most II+'s (see Appendix D). The two registers count through nine X positions and 10 Y positions, thus stepping through a total of 90 X,Y position pairs. The number in the X counter increments by one with each tick from a 90 kHz clock. After nine ticks, the X counter cycles back to zero and the Y counter is incremented by one. A complete cycle through all 90 possible X,Y pairs thus takes one millisecond.

The X lines are outputs from the AY-3600, and the Y lines are inputs. With each step of the 90 kHz clock, an electrical pulse is sent out on the one X line whose number is in the X counter. The AY-3600 then looks at the one Y line whose number is in the Y counter. When a keypress connects an X line to a Y line, then the pulse sent out on the X line output will show up on the Y line input. Most of the time this is not the case, and the X counter is advanced and a pulse is sent on the next X line. This continues until each of the nine X lines have gotten one pulse. Then the Y counter is incremented by one, and the cycle through the X lines repeats itself.

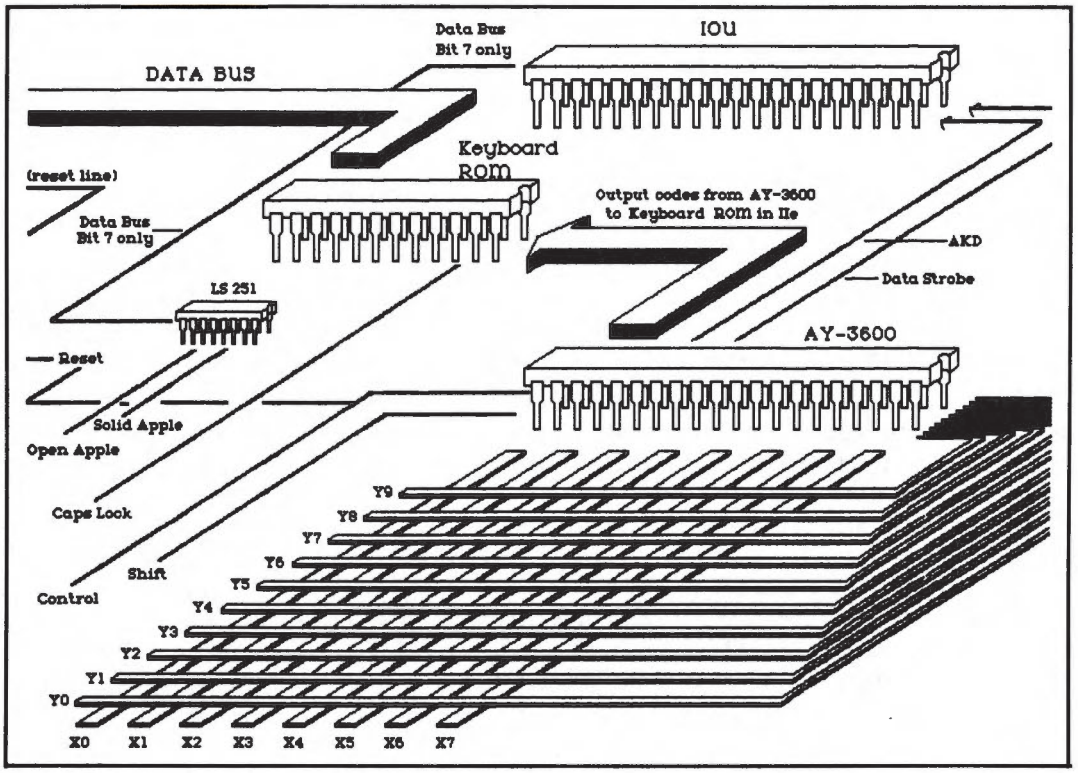


Fig. 8.2 The keyboard input system on the //e. The Open Apple and Solid Apple keys operate the game port push button lines 0 and 1. The layout is similar for the II+, but there is no keyboard ROM or AKD (any key down) line, and the data strobe goes directly to a data bus buffer.

Detecting a Keypress

If a key has been pressed, then at some time during the next millisecond the AY-3600 will send a pulse to that key's X line at the same time that the AY-3600 is looking at that key's Y line. At that moment, the pulse will appear on the Y input and the AY-3600 will know it has detected a keypress.

The AY-3600 immediately stops the advance of its X and Y counters, stops sending pulses, and changes into its analyse and report mode. The first active response is to turn on a signal called Any Key Down (AKD). This signal will remain on until the human has finished the entire keystroke and the key is no longer in contact. The AKD signal is used only by the keyboard electronics in the II and II+, but in the //e and //c it travels on into the IOU (Input Output Unit) and its status can be read at Apple address $C010 (decimal 49168; see Chapter 26).

Then, before proceeding any further, the AY-3600 comes to a complete dead halt for a period of eight milliseconds. This time is called the "debounce period" and helps the AY-3600 ignore any phantom making and breaking of the switched connection due to actual mechanical switch bounce (see Chapter 14). As the debounce period ends, the code translation step begins.

Translating the X,Y Position into an Output Code

The numbers that were in the X counter and the Y counter when the keypress was detected describe the X,Y position of the key that it is down. The AY-3600 must now match this X,Y pair with a digital code to send to the Apple. This chip has been designed so that any manufacturer can program into it which code will be sent for each possible X,Y pair. Although the same chip is used in the II+ and in the //e and //c, the AY-3600 translate program is different in the three machines.

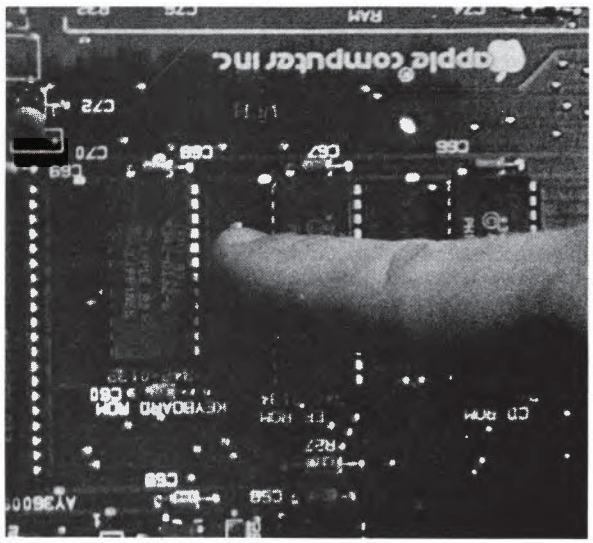
Before beginning the translation, the AY-3600 looks at two other inputs which are managed separately from the rest of the grid. One is from the Shift key and one is from the Control key. There are four possible Shift/Control situations (Shift, Unshift, Control and Shift-Control). Thus there are 90 × four = 360 possible output codes for an AY-3600. In the II/II+ only 47 of the 90 possible X,Y positions are active, and these are used with Shift and Control to generate 96 different output codes. In the //e there are 56 possible X,Y pairs and they are used with Shift and Control to generate 128 different output codes.

Another very important difference between the II/II+ and the //e and //c arises at this point. The AY-3600 output codes in the II/II+ are the actual ASCII codes which are passed onto the Apple data bus for use by the 6502. In the //e or //c, however, the AY-3600 output codes are just the first step in the translation process.

The //e and //c have a separate 2K ROM chip (see Figure 8.3) which receives the AY-3600 output codes and treats them as part of a ROM address. The ROM also receives an input from the Caps Lock key on the keyboard as well as from two additional signals on the motherboard. The //c keyboard ROM also gets an input from the "keyboard" switch. These various signals allow the ROM to alter the interpretation of the keys into 16 different possible layouts. This feature is used for some European keyboards, and it is used to simulate a II+ keyboard. Another interesting aspect is that it can respond as if the keyboard were in a Dvorak layout rather than a QWERTY layout.

The QWERTY layout of the keys dates back to the late 1860s and was designed to slow typists down so that they wouldn't swamp the mechanics of early typewriters. The Dvorak layout was designed in the 1930s to make it easier to type. It's design is based on selecting the 11 most frequently used keys and placing them all in the typists home row. It's been 50 years since the advent of Dvorak and it's never made a really big impact, but interested //e and //c owners should be happy to learn that their machines have an optional Dvorak mode.

Fig. 8.3a (right) The //e keyboard ROM permits reassignment of keys after decoding by the AY-3600.



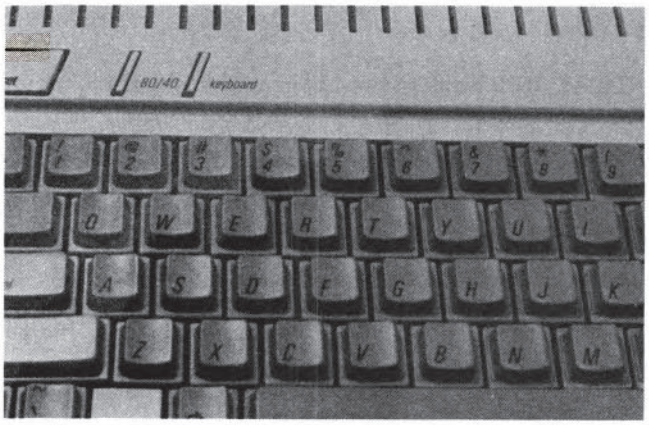


Fig. 8.3b (top left) The keyboard switch on the //c is connected directly to the keyboard ROM. When the switch is down, the ROM changes its responses to key codes from the AY-3600, thus converting the keyboard to a DVORAK arrangement.

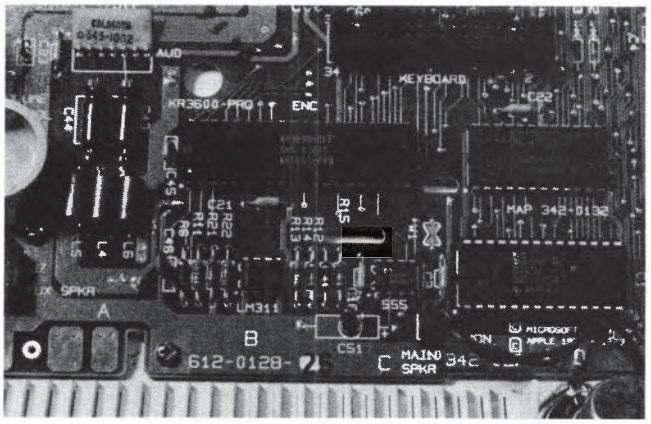


Fig. 8.3c (bottom left) The //c keyboard ROM is marked MAP. It receives and translates codes from the AY-3600 chip.

The Data Strobe and the Keyboard Flag Bit

Once the AY-3600 has selected the appropriate output code, it places that code in its output register and attempts to inform the 6502 that a new key has been pressed. When a program in the Apple looks at address $C000 (decimal 49152) it can see a seven bit character code from the keyboard. In the II/II+ this code is read directly from the AY-3600, while in the //e and //c it is read from the data pins of the keyboard ROM.

However, the eighth bit in the byte is not actually part of the character code. Rather, it is used as a signal from the AY-3600. There is a special output line from the AY- 3600 whose sole purpose is to set the eighth bit of $C000 to one whenever a new output code is generated. In normal operation, the AY-3600 waits about 11 microseconds after it has gotten its new output code ready to go, and then it turns on its Data Strobe line for a single 11 microsecond pulse. This pulse on the Data Strobe line is what sets the Keyboard Flag Bit to one.

Some further details of this Keyboard Flag Bit are discussed in Chapters 26 and 33. Standard protocol in the Apple requires that whenever a program reads in a character code from $C000, then that program must immediately do a "read" or "write" to address $C010 which is called the Clear Keyboard Strobe address. A read or write to $C010 clears the Keyboard Flag Bit to zero. The next time the program looks at $C000, it can check the eighth bit before it decides whether or not to do anything with the seven bit character code it sees there. If the eighth bit is set to zero, the program can assume that it has already read that character and thus ignore its continued presence in the input port.

In the //c, this keyboard Data Strobe line has a powerful additional function. It has a second connection to the Data Set Ready (DSR) pin of the 6551 communications chip used for serial port two. The actual purpose of this connection is to permit the AY-3600 to generate an interrupt when a key is pressed. This mode is used primarily when the //c is also using the modem port, but it can be used at other times to improve program performance. Most Apple programs spend the bulk of their time waiting around for a keypress to occur. In the //c, however, a program can do other useful work and attend to the keyboard only when an interrupt occurs. This function is explained in detail in Chapter 18.

Reviewing the Acquisition Sequence

In review then, when you strike a key, the AKD signal line turns on immediately, eight milliseconds later the AY-3600 has selected an output code to send, and 11 microseconds after that it sends a Data Strobe pulse to the Apple to set the Keyboard Flag Bit. Let's assume that the 6502 has been running around and around in its KEYIN program loop (see Chapter 33) continually checking the status of the Keyboard Flag Bit. A few microseconds after the AY-3600 sends the Data Strobe, the 6502 notices the change in the Flag Bit, grabs a copy of the character out of the input port, and then clears the Flag Bit back to zero.

This completes the sequence of detecting and acquiring a key press. When all this is finished, the output code is still sitting in the AY-3600 register, your finger is still just beginning the keypress (everything so far took just nine thousandths of a second), and the AKD line is still on.

N-Key Rollover

As soon as the AY-3600 has finished sending out its Data Strobe, it restarts its scanning activities. The X and Y counters begin to increment, pulses are sent successively on the X

lines, and new keypresses can be detected. Note, however, that this resumption has occurred long before the human has finished the previous keystroke. The AY-3600 will ignore any new report of the previous key until it has been released. Meanwhile, it will collect in as many new keys as you can press.

If you simultaneously push down all the keys on the keyboard, it will step through them one by one, stop the counter temporarily, decode, send Data Strobe and move to the next, always ignoring any that have already been reported. This is called "N-Key Rollover." It is important because it means that a fast typist who has several keys down in rapid succession before the first one is released will not confuse the keyboard.

Although the AY-3600 is equipped to do N-Key Rollover, the keyboard of the II/II+, as well as the //c keyboard, are not wired properly to permit the feature to operate correctly. When several keys are down, there begins to be a network of possible paths an X pulse can take on its way to a Y input. The result is that if you press more than two keys on a II/II+ keyboard, you will get phantom keys that have never actually been pressed. The //e has strategically placed diodes scattered about the grid switches which prevent these errant paths from developing. Thus the II/II+ and the //c have 2-Key Rollover, but the //e has N-Key Rollover.

II/II+ Repeat and //e and //c Autorepeat

Once a key has been pressed, its output code will remain in the AY-3600 output register until a different key is pressed, but once the 6502 has grabbed the code and cleared the Flag Bit, it will ignore its continued presence. However, if a new Data Strobe signal is sent, then the Flag Bit will be reset to one and the 6502 will think the key has been pressed again. This is the basis of the repeat function.

In the Apple II/II+, the repeat system is based on a "555 timer" which is set to an output frequency of 10 pulses per second. When the AKD signal is on and the Repeat key is pressed, the 555 timer is activated. Early Apple II's with the "single piece keyboard" (see Appendix D) used a keyboard decoder chip called the MM 5740, and that chip received the pulses from the timer and sent new Data Strobes. In the "two piece keyboard" used in later Apple IIs and all Apple II+'s there is an AY-3600 which has no built-in provision for a repeat function. In those Apples, the 555 timer is simply connected to the Data Strobe line directly.

The autorepeat system in the //e and //c is a bit more sophisticated. Recall that in the //e and //c, the AKD signal goes all the way down into the IOU. When the IOU sees the AKD signal turn on, it begins counting with an internal timer. After 0.9 seconds have passed with the AKD signal still on, the IOU goes into autorepeat mode. The Keyboard Flag Bit is actually a one bit storage location inside the IOU in a //e or //c, so it is no special problem for the IOU to begin resetting the Flag Bit on its own at a rate of 15 per second.

Some programmers who write games or who write for very young children may want to override this autorepeat function and so the //e and //c makes it possible to do so. Although the //e reference manual says that a read to $C010 gets you only the high bit, this address actually looks very much like $C000. You get the seven bit character code, but the eighth bit is set by AKD instead of by Data Strobe. This means you can ignore the usual strobe, clear, and repeat system and work directly from key down or key up information. You still get all the advantages of N-Key Rollover since the lower seven bits are updated with each new keypress even if AKD stays on.

Intelligent Encoding of the Output Codes

As noted, the //e and //c have a keyboard ROM placed between the AY- 3600 outputs and the Apple's keyboard input port. The only way a user can actually alter the codes is to push Caps Lock or the //c "keyboard" switch and there is no way to alter the coding process from within a program. Access to the various alternative coding bytes requires some cuts, solders and jumpers on the motherboard.

There are, however, a growing number of pieces of add on equipment for the Apple such as the Omega II keyboard from Zicor which can make this coding process much more elaborate. These "smart" keyboards can store hundreds or thousands of alternate character assignments. Most of these systems have their own supervisory microprocessor, RAM and ROM interposed between the keyboard decoder chip and the Apple's keyboard input port.

This happens to be one area where II/II+ owners are in much better shape than //e owners. In the II/II+, the MM5740 or AY-3600 is actually mounted on a special board beneath the keys (see Figure 8.4). The keyboard cable passively delivers the final ASCII code ready for use. This makes it easy to add extra electronics for intelligent decoding and simply feed finished ASCII codes on down to the motherboard.

In the //e and //c, however, the AY-3600 is mounted directly on the motherboard and what passes in the keyboard cable are the raw X line pulses and Y input return lines. For an intelligent keyboard system to operate it must not only generate the desired codes, but it must also simulate the appropriate switch closures to interface with the //e's AY-3600.

The only alternative is to unplug the keyboard ROM and intercede at that point, but future versions of the //e may have the ROM soldered to the board rather than in a removable socket connection, so designers have been hesitant to build products that require this change. This probably means that intelligent keyboards for the //e will require a regular expansion slot rather than plugging inconspicuously into the keyboard input port connector. In any case, this is one area where peripheral products for the II/II+ are in no way compatible with the //e.

Adding New Keyboard Features

The things you can do to improve upon your current keyboard situation include adding a numeric pad, adding a block of special function keys, or adding an entirely new detachable keyboard with a numeric pad and function keys built in. If you happen to be perfectly happy with all the things your keyboard does as it is, but wish it were detachable, then you can buy a fairly inexpensive case and cable from Innovative Micro Goodies or Keytec and follow their instructions for moving your Apple's standard keyboard out into a detached chassis. Both companies make II/II+ and also //e versions.

Another modest improvement many II+ owners might want to consider is the Repeaterrrr from High Order Micro Electronics. This is a small circuit board which plugs into your revision seven motherboard (see Appendix D) and adds autorepeat to all the keys on the keyboard. These folks also sell a Repeaterrrr + version which includes a little kit for making the Shift Key Mod.

The Shift Key Mod is a widely used trick whereby you connect your shift key to one of the pushbutton inputs in the game port. Many 80 column cards and word processing programs which can handle upper and lowercase characters will automatically check this input before

accepting each new character. The total effect is make the II/II+ shift key work like a real typewriter shift key. Figure 8.4 describes how to do the procedure yourself (see also Appendix D), but if you want to keep things as simple as possible you can purchase the modification ready to go from High Order Micro Electronics.

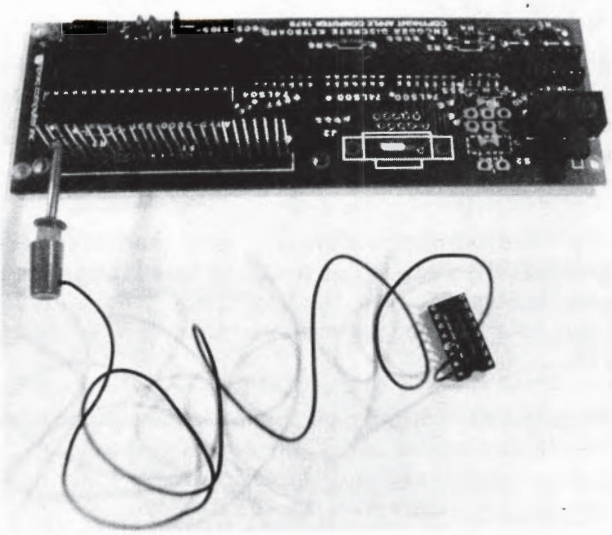


Fig. 8.4 The II+ keyboard piggy back board removed to show AY-3600. The switch on the right selects CTRL-Reset vs. Reset. The empty socket on the right is for the keyboard cable to the motherboard. The row of pins on the left connects to the actual keyboard.

For the Shift key mod, a micro test chip (Radio Shack pn# 270-370) is attached to the second pin from the left, thus tapping into the shift key signal. You can connect the clip easily with the top off your Apple and without actually removing the piggy back board. The wire from the test clip is then shoved into the push button 2 position on the game connector. (See Fig. 26.2c.)

Numeric and Spreadsheet Keypads

A compact convenient detached numeric pad is very handy if you're going to be entering a great deal of numeric data. You can get simple numeric pads for the II/II+ and //e from Apple and from Advanced Business Technologies. As explained earlier, the keyboard connector system for the II/II+ is very different than for the //e (see Figures 8.5 and 8.6), so you must specify which version you want. The ABT Keypad B is for the II/II+ and their Keypad K is for the //e (see Figure 8.7). If you have an old Keypad B, ABT will tell you how to modify and adapt it to work on the //e.

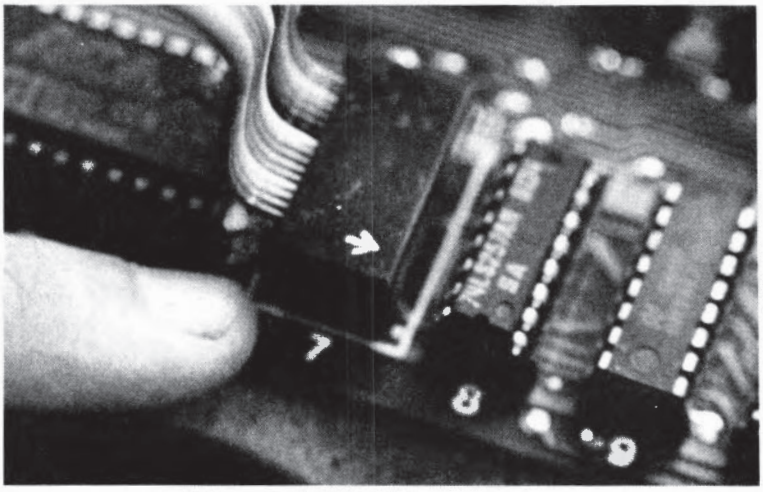


Fig. 8.5 The II+ keyboard connector. You may have to unplug the cable to install some keyboard upgrades, but always be sure to notice which way the plug is facing before you remove it.

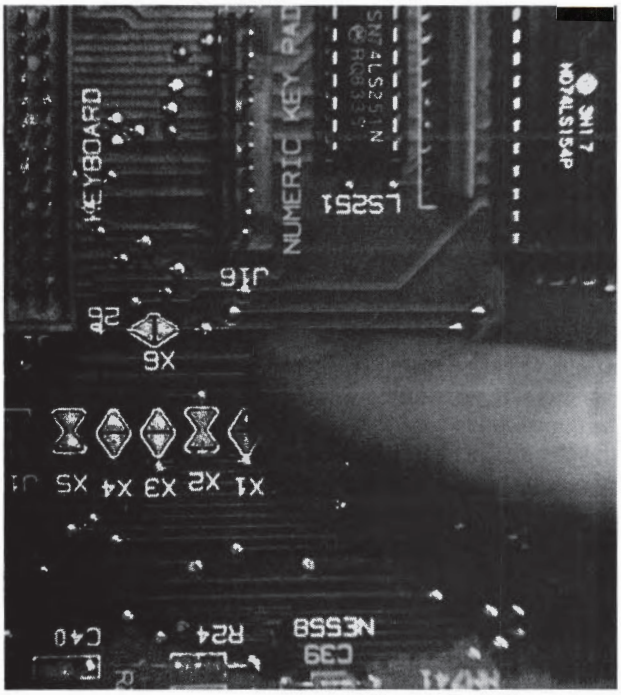


Fig. 8.6 The //e keyboard and numeric keypad connectors. If you use a slot 3 80 column board instead of a //e auxiliary slot card, you may need to do the shift key mod by putting a drop of solder on the "X6" jumper. The X6 jumper is already connected on Revision A //e's.

With a few more keys placed around the calculator style number keys, the pad can be considered a "VisiCalc pad." The Keywiz 83 from Creative Computer Peripherals has four cursor arrow keys, 15 VisiCalc command keys, and a shift key which assigns alternate commands to the 15 command keys. Thus you get 30 one or two stroke commands, cursor keys, and a numeric pad. The Keywiz 83 requires replugging the motherboard keyboard connector and it also requires a free expansion slot, but it is available for the II/II+ and for the //e.

Fig. 8.7 ABT 13 key numeric pad. Separate versions are made for the II+ and for the //e.

The //e Tender from Track House is a bit more modest, but is much more convenient than the Keywiz 83 for single handed use. It has a full 18 key numeric pad, a space bar, four arrow keys, and four programmable keys. These keys are programmed by setting switches and may be assigned to any single key you choose. The //e Tender is designed in conformance with the DIN design standards and so is a bit more comfortable to use than any of the other numeric or function key pads. The interface has been arranged so that you easily connect or disconnect the pad via a plug in the //e back panel. There is also a less expensive version in which the four keys are permanently assigned and there is a model for the II/II+.

Function Keypads

Videx, ABT, and Creative Computer Peripherals make systems which add intelligence and versatility to the normal range of functions of your Apple's keyboard. These products have to be considered in a cautious vein because you can get a fully detachable keyboard which does everything these systems do all for a comparable price. The Videx Enhancer II and Function Strip has been around for a long time and is comparatively inexpensive, but the CCP products are new and very expensive when compared to a full featured detachable keyboard.

There are two products from CCP that are arranged as function keypads. The Keywiz Convertible has 24 function keys, four cursor arrows and a shift key which alters the assignments of the function keys. You purchase the Convertible in a set configuration for any one of 10 different word processing packages. The Softkey from ABT is also a fixed configuration function keypad which offers either 15 Applesoft or 15 Pascal commands.

The second product from CCP, the Keywiz VIP (Very Intelligent Peripheral), has 31 function keys which can be shifted to 62 different outputs. There is also a select system which lets you switch among four complete sets of 62 keys. Further, this product has a special kind of memory chip called non-volatile RAM which permits you to key in your own assignments for each key. The RAM is called non-volatile because the VIP will remember your assignments even when the power is off. You can store up to eight characters for each function key and with a total of four $\times$ 62 equals 248 key assignments. Both of the CCP products can be used with either the II, II+ or //e.

The Enhancer II from Videx has a more limited market of the Apple II+ and later Apple IIs with the two piece keyboard, but it provides some attractive features at a low price. It is not a separate keyboard or keypad, but rather is an enhancement of the built-in keyboard electronics. You can also purchase a Function Strip which is a tape made up of 16 touch sensitive pads which you stick onto your Apple case above the keyboard.

The electronics serves to expand the possible character outputs to a full set of 128 ASCII codes and permits reassignment of any key on the keyboard. Videx has also developed a Dvorak keyboard option based on this system. You can store key sequences of up to a total of 510 characters distributed among the different keys. In addition, the Enhancer II can act as a 128 character "type ahead buffer" which means that if you are typing faster than your program's ability to accept characters, the Enhancer II will store up the extra characters and feed them on to your program one by one as it becomes ready for them.

Replacement Detachable Keyboards

There are five manufacturers of detachable keyboards for the Apple. All of the keyboards have numeric pads and some kind of function keys, but they differ quite a bit in the number of features provided. Of the five, only the EPS keyboard is available in a //e version.

Fig. 8.8 KB-200 keyboard from Keytronic.

The KB-200 from Keytronics is a fully DIN standard keyboard that closely resembles the IBM PC keyboard (see Figure 8.8). It has 10 function keys, but these are permanently assigned to a few DOS and Applesoft commands. It is rugged, a bit heavier than some of the other keyboards, but with a long enough cable for convenient use when sitting back, away from your desk. There is no provision for keeping the standard Apple keyboard active while the KB-200 is plugged in.

One alternative to permanently assigned functions is to leave all the function keys for programming by the user, which is the case on the MAK-II Accufeel from Multi-tech. This is a DIN style keyboard with 12 programmable function keys placed across the top of the keyboard. All of the other keyboards offer a much larger number of programmable and/or preassigned keys.

The Amkey Pro 100 is the more modest of the remaining three keyboards and is the only one of the five which lacks a cursor pad. It has seven permanently assigned DOS commands and 18 reassignable function keys. The function keys have three modes of operation. In mode 0 you can assign functions to the 18 keys, in mode 1 they offer an expanded set of Applesoft and DOS commands, and in mode 2 they provide 18 VisiCalc commands.

The most elaborate keyboards are from Executive Peripheral Systems (EPS) and from Zicor (the Omega II). The EPS board has 12 function keys which shift into four modes, giving you 48 active function assignments. However, this is an awkward board in two major respects. The physical layout is not optimal for reaching the function keys and there is no simple means of programming the function keys yourself. EPS sells a set of ROMs for various popular programs, but you can only plug in one of these ROM modules at a time, and all are at extra cost. EPS will make a ROM to order for you if you desire.

The best design of any of the boards is the Omega II (see Figure 8.9). It has 25 function keys and five built-into ROMs that support a large number of popular programs as well as a set for configuring your Epson printer. You switch instantly among assignments by tapping mode keys on the board. There is a sixth mode which allows you to program the keys yourself. The assignments are stored in 4K of static RAM in the keyboard but may be uploaded and downloaded to disk. With a single load you can have up to 31 characters assigned to each of the

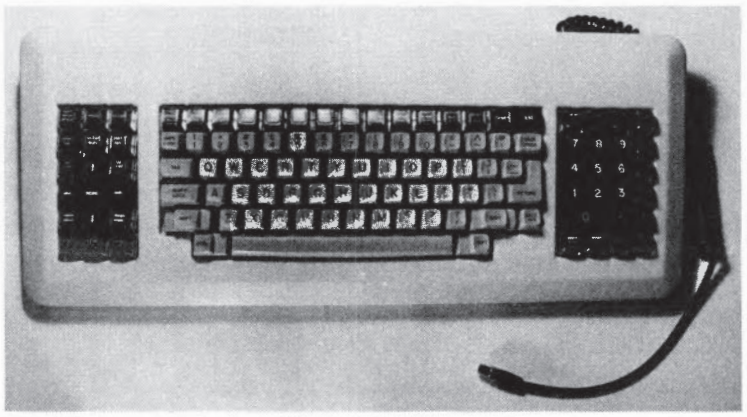


Fig. 8.9 Omega II keyboard from Zicor.

25 function keys, and 255 characters assigned to two additional long function keys. There is a full cursor compass with a home key in the center and these make up five more function keys also with six levels of assignment. Finally, there is a 128 character type ahead buffer which can be conveniently enabled or disabled.

The remarkable thing about the Omega II is that it is so incredibly packed with features yet it is beautifully laid out for convenient access to all features by a touch typist. The numeric pad is laid out as a 23 key VisiCalc pad and the cursor keys are on the left in reach of a touch typist's fifth finger. The board is light, rugged, reliable, attractive and has a 10 foot connector cable. Your standard keyboard remains active at all times, but there is a Caps Lock key (as well as a Shift Lock) which lets the Omega II simulate a standard Apple II keyboard.

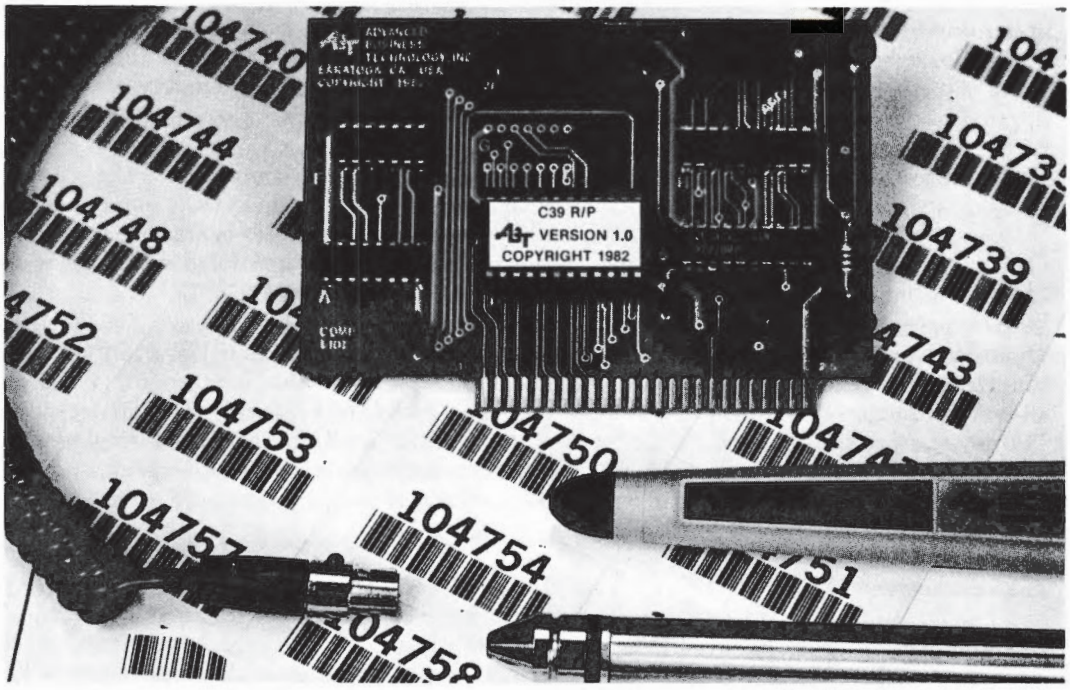


Fig. 8.10 Bar Code Reader.

Character Input without a Keyboard

The most exciting alternative to keyboard input is voice. A top of the line product such as the VIM from Voice Machine Communications (see Chapter 11) can respond to 80 spoken commands by recognizing your words and pumping the appropriate characters in through the keyboard input port. The VIM and a few other voice input products are described in the chapter on voice.

A far less dramatic but nonetheless very important character input device is the bar code reader. You can get the BarWand scanner from ABT as a separate device (see Figure 8.10), or you can buy it as part of a full scale "Retail Management System," complete with a cash register drawer, numeric entry pad, bar code printing software, and point-of-sale/inventory management software (see Figure 8.11).

Fig. 8.11 Retail Management system.

Chapter 9

Spatial Input

Spatial Input

The physical designs of spatial input devices draw on three rather distinct aspects of our normal human expression and action. Pointing is natural form of human gestural communication, drawing is a more attentive means to the indirect communication of art and image; hand control is built on our enthusiasm for operating mechanical devices (buttons, knobs, levers and triggers).

Any one of the many major distinct kinds of spatial input devices can be made to do the tasks of all of the others. However, each emphasizes some particular aspect of our abilities for spatial expression. Digitizing tablets and digitizing cameras stay closest to drawing and help maintain the exact physical sensation of using a pen or paint brush. To achieve this simulation, they require the most elaborate electronics and mechanical equipment and hence are much more expensive than other spatial input devices.

Joysticks, trackballs and game paddles are built to emulate our favorite mechanical control devices and emphasize force, orientation and directional attention. The mechanisms involved tend to be simple and inexpensive, but they provide very little of the sensation of drawing or pointing. The future descendants of these devices may include the equivalent of micro manipulator or macro manipulator gloves which detect and scale detailed motions of the hand. This sort of thing is not too important for playing games, but could be very useful for programming robot devices.

The major competitors for the best pointing and drawing device are the touch screen, light pen, touch pad and mouse. Of all of these, the mouse is gaining the most rapid acceptance among microcomputer users in part because it is comparatively inexpensive, but also because it comes closest to meshing with our feel for all of pointing, drawing and mechanical hand control.

The Touch Screen and Light Pen

The most direct analogy to drawing and pointing to objects on paper is achieved by the touch screen and light pen which let you interact directly with the image on the CRT. A touch screen such as the CTA Touch Bezel is based on a frame which mounts on the front of your monitor and which sets up a grid of infrared light beams and photo detectors. Touch screens first came into heavy use by air traffic controllers during the 1960s. The CRT screens got messy and dirty and people's arms got tired.

Nonetheless, for intermittent use where precise pointing isn't necessary, these can be very handy. The screen system from Computer Technology Associates has a resolution of 96 horizontal points by 64 vertical points so it should be possible to select any one of the characters on an 80 by 24 column text display. However, fingers are thicker than characters. To take full advantage of the precision, you need to use some narrow pointer to break the beam which defeats the whole idea of spontaneous pointing. The optimal use is to permit casual users to select menu options by pointing to large, well labeled blocks on the screen.

A light pen is actually just a fancy, hand held photo detector. As explained in Chapter 5, the image on the CRT screen is created by an electron beam which sweeps through a "raster" pattern on the screen (see Chapter 5, Figure 5.1). The beam draws a thin horizontal line, then snaps back diagonally to the side it started on but pointing a little lower in preparation for drawing the next horizontal line. A complete coverage of the screen takes 1/60th of a second. The beam is always "shining" as it draws, but it must emit a much brighter burst to make a phosphor glow as a dot on the screen.

When you point the light pen at a position on the screen, it waits until it sees the beam sweep by and then sends a signal to the computer. The Apple directly controls the timing and position of the electron beam sweeps, so it is a fairly straightforward task to match up an absolute screen position with the time at which a pulse arrives from the light pen.

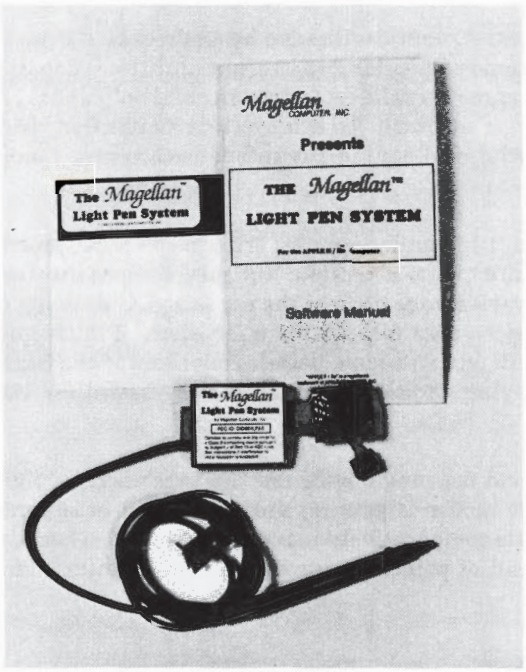


Fig. 9.1 Magellan light pen. The light pen will allow the user to interact directly with the image on the CRT screen. The resolution of a light pen can be much better than a touch screen.

The resolution of a light pen can be much better than a touch screen, and both Magellan and Gibson Laboratories claim that their pens can pick out a single dot on the Apple high res screen. The Magellan pen collects up its video timing information from the internal video connector on the Apple motherboard and reports detection via one of the pushbutton inputs on the game port (see below). This light pen also has a button on its side (see Figure 9.1) which you can use as a secondary select button much as you would use a button on a mouse.

The strong point of the simple interface used by Magellan is that it helps keep down the cost of the system. However, it does place a heavier burden on the programmer for interpretation. The Gibson light pen is plugged into slot 7 and uses special video synch signals provided on the pins of that slot. The extra electronics on the Gibson card makes this system easier to deal with for most programmers, but it also makes the system considerably more expensive. Another problem is that early Apple //e's with a revision A motherboard (see Appendix D) don't have the necessary video synch signals on slot 7.

There are a few other limitations with both light pens. First, it takes a bit of practice to get used to pointing the pen at the right angle for good detection which further detracts from spontanteity of use relative to other pointing devices. Then, because the pen is light and held at a distance, there can be trouble with jitter for high resolution work. In addition, you should note that these pens don't work with amber screen monitors nor with older Monitor IIIs which have a long persistence phosphor.

Touchpads and Mice

To use a touchpad or a mouse you have to maintain a sort of mental image of a secondary map of the screen. You point to a surface on a desk top but the action happens elsewhere on the CRT. Remarkably enough, this proves to be a fairly effortless task for most people. For instance, our seven-year-old reviewer immediately latched onto the Koala touch pad and happily used it for hours on end. A touch pad is small and convenient and unlike a light pen or touch screen there is no problem with fatigue from holding your arm up and your hand doesn't obscure the image on the screen.

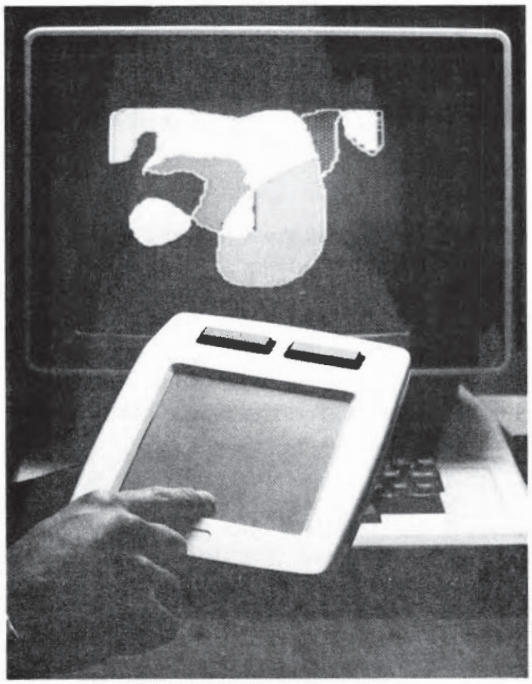


Fig. 9.2 Koalapad Touch tablet.

The KoalaPad touch tablet (see Figure 9.2) is a low resolution graphics tablet which plugs into the Apple game port and which can detect and follow the press of a finger without the need for a special pen or marker device. Like a touch screen, the resolution you get when drawing with a finger isn't too good, so you often end up using a stylus. It's impossible to remember exactly where you last put your finger down so you always have to first make contact and then trace to where you left the cursor if you want to draw smooth lines or trace to the menu box you mean to select. The pad has two buttons for confirming your choice of a location and for controlling software options.

To use a mouse, you must have a square foot or so of space clear on a clean desk top in easy reach when you're working. Some people don't work like that and may want to use a KoalaPad, but there are several advantages of using a mouse. First, the mouse stays put when you take your hand off it so you eliminate the initial contact and search step. More importantly, it is convenient to design a mouse which provides good manual control at very high resolution.

The Apple II Mouse

The mouse system for the Apple II from Apple Computer is extremely complex and sophisticated. It has a delightfully fast and precise response to movement, and is extremely easy to use. However, that speed and precision required a remarkable amount of skillful hardware and software engineering. The AppleMouse interface card for II, II+, and //e computers actually has a 6805 "microcomputer on a chip" to carry out most of the work. The 6805 has its own RAM and ROM inside the DIP with it, and thus it can monitor the mouse and store information about movements without any effort on the part of the 6502.

In the //c, however, all of the work is done by the Apple's own 65C02. This involves an elegant, multilayered system of precisely timed interrupts and high speed interrupt handling routines. You have to be careful of your information source if you're trying to learn about the system since several small but crucial errors managed to creep into the description in the first release of the //c technical reference manual.

The Mouse Itself

The Apple II mouse differs from the Lisa mouse in that it is designed to work in a smaller area of desktop space. As you track in one direction, there are 45 clicks per inch on the Apple II mouse, but just 31 clicks per inch on the Lisa mouse. At a density of one click per screen pixel, you scan 280 dots in just 6.2 inches or 560 super high res dots in 12.4 inches. Lisa needs at least 23.2 inches for a full sweep of its 720 dot horizontal screen. In fact, as you will see shortly, there is not a strict linear relationship between distance moved and dots counted, but these measures make it fairly clear that you can, for instance, roll the mouse around on the cover of a paperback novel lying near your chair.

Inside the mouse, there is one switch detector for the button and four "optical detectors" for monitoring motions. Each optical detector involves a small wheel which sits in contact with the mouse's large rubber ball. When you push the mouse across a desktop, you cause the rubber ball to roll, and it turns the small wheel.

There are little square holes around the circumference of the wheel and there is a tiny Light Emitting Diode (LED) trying to shine through one of the holes to activate a light detector on

the other side of the small wheel. If you were inside the detector, you would therefore see the light source blink on and off as the wheel turned. The light detector is connected to the Apple's mouse interface, so this lets the Apple watch the wheel. When the mouse is still, the detector sends a steady signal, but the signal begins to blink on and off as the mouse moves.

X-MOVE and Y-MOVE

If you move the mouse in one direction at a steady speed, one of the optical detectors will thus send a steady square wave signal to the Apple. As a new hole moves into register, the signal rises, then, as the hole passes, the light beam is cut and the signal falls. The Apple usually attends only to the "rising edge" of the signal, although it can be configured to watch only the "falling edge" of the signal instead. The //c softswitches at $C05C through $C05F (49,244 to 49,247) are used to select rising or falling edge.

One of the detectors is called X0 or X-MOVE and it is positioned in the mouse casing to turn when you move the mouse to the right or left. A second one called Y0 or Y-MOVE is placed at right angles to the first one, and it detects forward or backward motion. With these two detectors you can find out when the mouse is moving, how fast it is moving, and even which axis or axes it is moving on; unfortunately, you can't tell which direction either wheel is turning in.

X-DIR, Y-DIR and Interrupts

To determine direction, Apple added two more optical detector wheels. The first of these is called X1 or X-DIR and it is placed exactly on the other side of the rubber ball from X0. When you move the mouse to the right or left, both wheels turn in unison. However, the two wheels have been installed so that their holes are a little bit out of register. What this means is that when you roll the mouse to the left, the signal from X1 always turns on just before the signal from X0 (and, of course, turns off first as well). When you roll to the right, X0 always turns on first, a few instants before X1 turns on.

To detect motion and direction, the Apple watches the X0 signal line. Whenever it sees a rising edge, it immediately goes about seeing whether or not X1 is also on. If so, the mouse has just moved one click to the left. The Apple //c must check X1 within about 50 microseconds (millionths of a second) to be sure it gets this direction information correctly. A similar situation applies for Y0 and a signal called Y1 or Y-DIR. The Y1 signal will be on if the mouse has moved backward, and off if the mouse has moved forward.

Obviously, this system will fail miserably unless the Apple is paying very close attention to the X0 and Y0 lines at all times while the mouse is being used. Not only must it notice the change on X0 or Y0, but it must also be prepared to check immediately to learn the status of X1 or Y1.

The only way to accomplish this is with a high speed "interrupt" system. When such a system is working, a signal such as X0 is able to activate a switch inside the 6502 microprocessor which halts whatever program is in process. The 6502 then goes about servicing the interrupt request and, when it is done, goes back to what it was doing before.

The AppleMouse Interface Card for the II, II+ and //e

Unfortunately, the Apple II, II+ and any //e's sold before the summer of 1984 cannot perform this sort of interrupt properly (see Chapter 27). This is why it was necessary to put a 6805 microprocessor on the AppleMouse interface card. The 6805 devotes its full attention to the

mouse, and carries out leisurely information exchanges with the 6502 at convenient times. Any Apple //e can be upgraded to handle high speed mouse interrupts (this requires replacing some ROMs), but because of the 6805 in the AppleMouse interface, most users will find they don't need the upgrade just to use a mouse.

If you are using DOS 3.3, Pascal or CP/M, then your programs have to pause periodically to interrogate the 6805. The 6805 has 128 bytes of RAM which it uses to store current data on mouse movements. It takes care of the fast check of X1 and Y1 and uses the data to calculate, update, and store an absolute X,Y position as well as to watch for presses of the mouse button. However, storage space is very limited, so if the mouse is moved rapidly, your program may not interrogate the 6805 often enough. As a result, this "passive" mode may allow some loss of information.

VBL Interrupts with ProDOS

The ProDOS operating system makes it possible for any Apple II to handle certain kinds of less urgent interrupts. Therefore, when the AppleMouse interface card is being used with ProDOS, its 6805 can be configured to generate interrupts at a steady rate of 60 times per second.

This sort of 60Hz interrupt can be very important in an Apple. Those of you who have worked through the section on video will know that 60 times a second the Apple launches into a complete "refresh" of everything it has put on the video screen. However, there is a period of time just before each refresh during which the electron beam inside the CRT is effectively turned off, and that period of time is called the Vertical Blanking Interval (VBL). During VBL, the 6502 can rearrange things in the Apple's "display memory," without disrupting what you actually see on the video screen.

In fact, one of the great frustrations for game programmers on the Apple II and II+ was that it was impossible for a program to find out when a VBL was going on. In the //e, a program can find out when a VBL is occurring by looking at location $C019 (49,177), but this means the program has to hang out in a wait cycle, checking $C019 again and again until it finally reveals a VBL. However, if you have an AppleMouse Interface Card installed and ProDOS running, you can use the 60Hz interrupt as a "VBL interrupt". The interrupts are, in fact, locked into the video timing, so it is a true VBL interrupt.

To understand how the mouse system could work with VBL, think of a program that wants to move a mouse cursor around on the screen. During each VBL, the 6805 sends an interrupt request (IRQ) to the 6502. The 6502 responds by interrogating the 6805 about mouse position and status, it then moves the mouse cursor in display memory, and when the next video refresh begins, the mouse will have moved. At a rate of 60 mouse interrupts a second, the mouse will seem very responsive to your average human user, and there will be no lags between movements of your hand and cursor movements on the screen.

The //c, the Mouse and Interrupts

Unlike older Apple models, the //c is capable of operating as a high strung, ultra-responsive, interrupt driven machine. In older Apples, whenever something happened outside the machine such as a keypress on the keyboard, the arrival of a new byte from a modem, or a movement of the mouse, the information had to be held in a temporary storage buffer until the Apple got around to checking all the various buffers. This sometimes meant that information got

lost, but it almost always meant that the Apple was set up to pay attention to only one source of outside information at a time and that it was always wasting time in waiting loops, checking and rechecking certain buffers, hoping for something to happen.

The //c can respond to interrupts from either one of the two serial ports, from the keyboard, from a device connected to the external disk drive connector, from its own VBL interrupt generator (see above), and from the X0 or Y0 line from the mouse. All of these different "interrupt sources" lead to a common pathway. There is just one IRQ (interrupt request) line leading into the 65C02 and all of them must share it. Therefore, when the 65C02 detects an IRQ, its first task should be to figure out which source the interrupt came from.

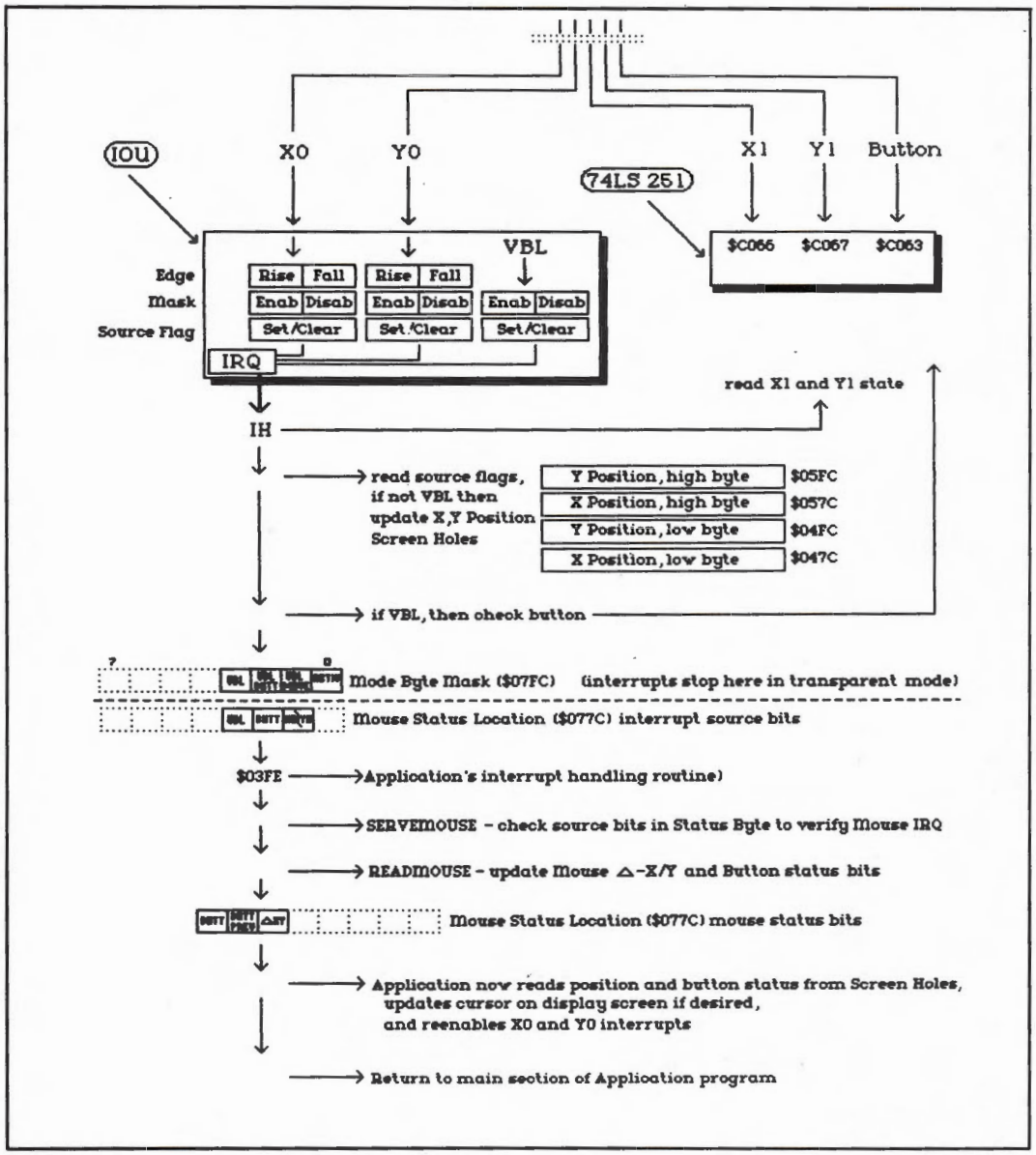


Fig. 9.3 Overview of //c Mouse interrupt system.

//c Interrupt Source Flags

To make the system work, each interrupt source requires both a connection to the IRQ line and a little bit of RAM where it can set up a flag. The source sets up the flag and then pulls the IRQ line. The 65C02 responds by scanning through the various "interrupt source flags" until it figures out which source wants attention.

In actual fact, because of the extremely demanding time requirements of the mouse X and Y direction system (see above), the 65C02 always starts by reading X1 and Y1, no matter who generated the interrupt request. Later, if it finds out that the IRQ was due to a movement of the mouse, it looks back at the X1 and Y1 data it grabbed and starts calculating the mouse position.

In summary, in order to function properly, the //c interrupt system must always be prepared to respond immediately to an IRQ, it must always first check X1 and Y1 before 50 millionths of a second have gone by, and then it must scan through the interrupt source flags to figure out who called.

A Sure, Quick Start

The Apple has a complex memory bank switching system (see Chapters 25 and 26), so it must be prepared to act correctly no matter how the memory is set up at the moment the IRQ arrives. The details of the 65C02's response to IRQ are covered in Chapter 27, but the principal result is that the 65C02 reads the numbers stored in the two highest bytes in memory (addresses $FFFE and $FFFF; 65,534 and 65,535). Those two bytes tell the 65C02 where Apple Computer has stored the Interrupt Handling routine, and it acts on that information to find the routine and start running it.

Normally $FFFF and $FFFE refer to permanent locations in the Apple's Monitor ROM, but in a //c with a running mouse the contents of those two locations are copied into all of the various chunks and pieces of RAM which can sometimes find themselves switched into the address space. Therefore, no matter what the memory configuration when the IRQ arrives, the 65C02 can always find the address of the interrupt handling routine.

However, the Interrupt Handling (IH) routine itself might be switched out of the address space somewhere. In calmer circumstances, the 65C02 could probably use some short routine to switch all the memory back to a standard configuration, and then look for the Interrupt Handler. But remember, there's only 50 microseconds before X1 and Y1 may become invalid. Fortunately, in the //c, the entire 4K range of the $C000 space is permanently locked into the address space (see Chapter 22). Therefore, the Interrupt Handling routine is kept at $C803, permanently in the address space.

(This is one point where it is very difficult to upgrade a //e to give it the features of a //c, because in the //e there is no area of permanently locked in ROM. In the mouse interrupt upgrade, the //e interrupt handler is put in with firmware for the //e 80 column system ($C300). This is done because the $C300 page is the closest thing a //e has to permanently active ROM. Videx has a specially modified version of the Ultraterm to accommodate this.)

Mouse Inputs and Interrupts

To monitor the mouse, the //c must watch five different signal lines: X-MOVE (X0), Y-MOVE (Y0), X-DIR (X1), Y-DIR (Y1), and BUTTON. The X1, Y1 and BUTTON signals are handled just as if all three were just buttons on a game control. In fact, BUTTON actually is connected to the PB2 push button input ($C063) which is built into all Apple IIs. The X1 and Y1 lines are connected in the same way as BUTTON, but they replace two of the analog game paddle

inputs (PDL2 and PDL3, $C066 and $C067) which are built into older Apple IIs. These three inputs are connected to a 74LS 251 (see Chapter 13) and all are read by looking at their address ($C063,$C066, or $C067) just as if they were all push buttons.

The X-MOVE and Y-MOVE signals however have a very different input system. They are connected directly to the Input Output Unit (IOU), where they may be used to generate interrupts. In the //c, the IOU has an IRQ output which it can trigger in response to its built in VBL timer, or in response to a signal from X0 or from Y0 (see Figure 9.3).

This section of the IOU is new in the //c, and it involves a number of new softswitch control locations (see Chapter 26). For each of X0 and Y0 there is a pair of switches to select whether the IRQ will occur in response to the rising or the falling edge of the signal from the mouse (see above). These are usually set to monitor the rising edge for both X0 and Y0 ($C05C and $C05E).

The next level of control in the IOU is to do with whether or not the incoming signal will be allowed to actually generate an IRQ. Both X0 and Y0 are controlled by a single softswitch pair:

$C058 = Disable X0/Y0 IRQ
$C059 = Enable X0/Y0 IRQ

and there is a second pair that controls VBL interrupts:

$C05A = Disable VBL IRQ
$C05B = Enable VBL IRQ

Aside from the obvious purpose of shutting off the mouse when it's not being used, these "enable/disable" switches get heavy use within the built-in firmware mouse routines.

When an appropriate edge arrives, and if that interrupt source is enabled, then an "interrupt source flag" must be marked and the IRQ signal turned on. The three interrupt source flags in the IOU can be read as follows:

$C015 = Read X0 interrupt source flag
$C017 = Read Y0 interrupt source flag
$C019 = Read VBL interrupt source flag

Some of these locations are identified incorrectly in some //c technical literature from Apple. $C015 and $C017 read but do not reset interrupt source flags.

After the interrupt handler has responded to the IRQ, scanned through the flags and located the source, it has to be sure to clear or reset whichever flag was set, thus avoiding confusion during the next interrupt. A read or write of $C048 resets both $C015 and $C017, no matter which one was set, and a read or write of $C070 resets VBL.

This completes the picture of the hardware and the softswitches involved. To review, there are three real sources for interrupts from the IOU: X0, Y0 and VBL. The VBL signal comes from within the IOU, while the X0 and Y0 signals are caused by moving the mouse. If one of these sources is enabled, then it is capable of setting an interrupt source flag in the IOU and causing an IRQ signal to be sent to the 6502.

//c Firmware for Managing Mouse Interrupts

The various machine language subroutines that operate the //c mouse interrupt system fall into three categories. First is the master //c Interupt Handler (IH) itself which actually does most of the work. Next is a collection of mouse firmware routines which are available for use by any machine language program wishing to use the mouse. Finally, there must be a third level which is the applications program such as MousePaint or your own program.

When the IH is finished with its chores, it turns control over to the application. At this point, the application must have its own interrupt subroutine which follows a special protocol prescribed by Apple. The application uses the mouse firmware routines to collect information assembled automatically by the IH.

The Status and Mode Locations

In order to provide for communication among these three categories of subroutines, Apple has reserved two RAM memory locations which can be called the status and the mode locations. The two locations are in "screen holes" in text page one (a screen hole is an unused location in RAM stuck in the middle of hundreds of locations used for managing the video display; see Chapter 5 and Chapter 21, Figure 21.6b). The address of the the status location is $077C and the address of the mode location is $07FC.

The status and mode locations act in many ways like the hardware in the IOU for interrupt source flags and for IRQ enable/disable (see above). The IOU stands between the mouse on the outside, and the 65C02 on the inside. If IRQs are disabled in the IOU hardware, the 65C02 will never be aware of mouse movements. Similarly, the status and mode locations stand between the automatic, high speed IH on one side, and the particular applications program on the other. Mode locations can be used to let the IH do its work "transparently" without the applications program ever becoming aware that an interrupt has occurred. The status location plays the part of the interrupt source flags. The Applications program learns what kind of interrupt has happened by looking at the status location rather than by looking at the IOU flags themselves (see Figure 9.3).

Mouse Work Done by the Interrupt Handler

Whenever an IRQ arrives at the 65C02, the IH quickly does some housekeeping in and around the 65C02, and then begins almost immediately to read the X1 and Y1 inputs in order to preserve the very volatile direction information they may contain (see above). Next it identifies and records the current memory configuration and then switches everything to a standard state. It is now ready to begin its search for the interrupt source flag, and the first place it looks is to the IOU.

If the interrupt was caused by a movement of the mouse, it determines if X0 or Y0 was responsible, and then uses the X1 or Y1 direction information it is holding to begin to calculate a new absolute position for the mouse. The mouse position is stored in four X,Y Screen Hole locations:

$047C X - low byte
$04FC Y - low byte
$057C X - high byte
$05FC Y - high byte

The Interrupt Handler actually updates the absolute position at this time and stores the new value in the appropriate screen hole locations. If any "clamping limits" (see below) have been set, it is the IH that takes these into account at this time.

The IH can finish up by resetting the X0 and Y0 interrupt source flags to 0, and then reactivating the program that was running at the time of the interrupt. This is the "Transparent Mode" of operation. The user's program can look at the X,Y Screen Holes at any time to find out where the mouse is, but will never know that any of the interrupting or updating has occurred unless it looks.

The Transparent Mode presents the modest advantage of being able to function with just about any program, including your own written in Applesoft BASIC, without any requirement for a special mouse handling routine. However, in this mode, there is no way to be sure of noticing a button press, nor is there any way to ensure smooth tracking of a mouse cursor on the screen.

Using VBL to Enhance Mouse Performance

Most programs written specially to use the mouse will operate it in what is called Movement-Button Interrupt Mode (MBI Mode). In this mode, the applications program is interrupted automatically 60 times a second so that it can attend to details of following the mouse accurately. When this mode is active, the Interrupt Handler does additional work to gather and present mouse information and then passes control to the application program's interrupt subroutine rather than just reactivating the application in the middle of what it was doing before the interrupt. Thus, the applications program can gather up all the information collected by the IH, and take care of such details as moving the mouse cursor on the screen or accepting a selection from a menu.

In MBI mode, when an X0 or Y0 interrupt occurs, things proceed just as in Transparent Mode all the way through to the point where the IH has just updated the X,Y Screen Holes and reset the X0 and Y0 interrupt source flags (see above). Now, however, the IH takes a few steps to make additional information available to the user's program. First, it disables all X0 and Y0 interrupts. This means that all further movements of the mouse will be missed until after the user has collected the information on position and reenabled X0 and Y0 interrupts. This is the responsibility of the user's program.

In addition, the IH records a bit called Delta which simply signifies that there has been some movement in X or Y. This bit is stored temporarily in a screen hole at $067C. At this time, however, the IH does not report the interrupt to the user's program. Rather, it leaves the Delta bit set, and finishes up just as in Transparent Mode.

The next interrupt to come along cannot come from X0 or Y0 (these are still disabled), so it will most probably come from VBL. When the VBL IRQ occurs, the Interrupt Handler discovers that fact by looking at interrupt source flags, and the first thing it does is to look at the Delta bit in $067C. If it has been set, then IH goes about reporting the fact by putting an appropriate bit into the Status Location at $077C.

Recall that the user's program will use this status location as a sort of interrupt source flag. There are three interrupt source bits in the Status Location. One of them is called X0/Y0 since it is set when either an X or Y move has occurred. A second one is called Button because it will be used to simulate button interupts, and a third is for reporting VBL interrupts (see Figure 9.3). In the sequence described above, only the X0/Y0 bit is set.

Now that the IH has done the software equivalent of setting an interrupt source flag, it does the software equivalent of sending in an IRQ. That equivalent is a jump through the "interrupt vector" at $03FE (see Chapters 21 and 27). This is how the IH activates the application program's interrupt subroutine. For now, however, before looking at what the application's subroutine must do, it's worth saying a little bit more about VBL and the "simulated button interrupts" just mentioned.

Simulated Button Interrupts

Since a VBL interrupt occurs 60 times a second, the IH will often look at the Delta bit in $067C and find that there has been no change in X or Y position. In this case, IH checks to see if the mouse button has been pressed (by reading $C063). If it has, then IH will set the appropriate bit in the Status Location at $077C and then do a jump through $03FE to activate the application program's interrupt subroutine.

From the point of view of the application program, this is just the same as if its interrupt subroutine were called after an X or Y move. The way that the application program determines the cause of the jump through $03FE is to look at the Status Location. There, it will find a source bit set either for an X/Y move or for a button press. Thus, the application program finds itself in a situation in which it is interrupted AND its interrupt subroutine is called during some VBL intervals, but never more often than 60 times a second. When this occurs, it will discover evidence of either an X/Y move or a button press.

There is another mode in which a jump is done through $03FE everytime a VBL interrupt occurs, whether or not anything has happened with the mouse. This will be useful only for some graphics-intensive programs that always need to know about VBL. In that mode, it will find one of three conditions in the Status Location: evidence of an X/Y move, evidence of a button press, or evidence only of a VBL interrupt source.

Mouse Firmware and Applications Programs

Once a jump through $03FE has occurred, the applications program is responsible for managing a fair amount of bookkeeping. To simplify things, Apple has provided a small collection of mouse firmware routines which the application should use for interacting with the IH.

First, the application should call a routine called SERVEMOUSE which does nothing but look at the Status Location and report if any one of the three Interrupt Source Bits (X0/Y0, button, VBL) have been set. If it comes back with the carry bit set, then it found something, and you're supposed to call READMOUSE.

READMOUSE doesn't actually read the mouse inputs or update the X,Y coordinates or anything so impressive. All it does is to look around a little and come back with a simple report of mouse status. This mouse status report involves the setting of three Mouse Status Bits. These three bits are kept in a different part of the same Status Location ($077C) that holds the three Interrupt Source Bits (see Figure 9.3). The three bits are: button up or down, button up or down on previous call to READMOUSE, and X/Y Move.

At this point the user's program can go about reading the X,Y position from the X,Y Screen Holes and deciding what it wants to do about the button status. It must also reenable X0 and Y0 interrupts at the level of the IOU ($C059), but this should not be done until after retrieving the new X,Y coordinates from the screen holes.

The other mouse firmware routines available for the user's program are INITMOUSE which sets up all the screen hole locations and softswitches, and SETMOUSE which is used to activate

the mode you want (transparent, movement interrupts, button interrupts, movement-button interrupts, movement interrupts with full time VBL, etc.). In addition, you get CLEARMOUSE, HOMEMOUSE, POSMOUSE, and CLAMPMOUSE for such tasks as homing your mouse cursor or setting up window boundaries.

Finally, the one other important resource to keep in mind is the //c MouseText Character set which lets you move graphic characters around on the text screen. These are text characters from the point of view of the scrolling and video system, but they give you useful shapes for cursors, window boundaries, and icons. These can be installed in a //e as well (see Chapter 26 for more details).

Hand Controls for Games

The internal game port connector in the Apple II, II+ and //e has provisions for seven input signals and five outputs. The outputs are described in Chapters 14 and 26, and they don't get used for feedback by any of the popular hand control devices. These outputs are not available on the external game connector on the //e, and they have been removed completely in the //c.

In the II, II+ and //e, there are four "analog" or "game control" input ports (called GC0, GC1, GC2, and GC3 in the II/II+ manual and called PDL0, PDL1, PDL2, and PDL3 in the //e manual) which can be used to measure the position of a knob or lever. In the //c, PDL2 and PDL3 have been removed to make room for the X-DIR and Y-DIR inputs from the mouse (see above discussion of the mouse). The detailed operation of the analog input system,described in Chapters 14 and 26, allows for detecting 256 steps of position. The detection and measurement process is fairly rapid, but varies with the distance from the zero setting. When you're all the way over near zero, your position is confirmed about once every 15 millionths of a second while at the other extreme you still get checked every three thousandths of a second, so that, in any case, you are not likely to move too fast for detection.

The remaining three inputs (PB0, PB1 and PB2) are simple pushbutton detectors. Many Apple II/II+ owners have connected a wire from their shift key to the PB2 switch input to simulate a regular shift key (see Chapter 8). Many word processing programs and 80 column card firmware routines check the status of PB2 every time they read a character. For these reasons, games and game controllers tend to stay away from PB2. In the //c, PB2 also serves as the connection point for the button on the mouse. In the Apple //e and //c, the Open-Apple key is connected to PB0 and the Closed Apple key is connected to PB1. Both PB0 and PB1 are used in many games, and this means you can do your firing directly from the //e or //c keyboard if you like.

Most programs that use hand controls treat PDL0 and PDL1 as measurements of an X axis and a Y axis respectively and then group PDL2 and PDL3 together for use by a second pair of inputs, i.e., for a second player. In a Game Paddle you control only one axis, thus you need two game paddles to get full control of the X and Y axis, and you need four game paddles for two players each to control both X and Y. In part, this dates back to the great grandfather of most video games, Space Invaders, in which you moved from side to side behind barriers along the bottom of the screen and used the attached pushbutton to fire, but in which you never had to move forward.

Game paddles from Kraft (see Figure 9.4), Apple and TG all emphasize precise positioning of the rotary knob which sets the position of a single axis. Many games are based on this kind of input, so paddles are still quite popular. You choose a paddle either based on the precision

of the rotary knob or on how the thing feels in your hand. The Apple paddle emphasizes forceful operation of the firing button with the thumb, while the TG game controller features easy reconfiguration for left handed operation.

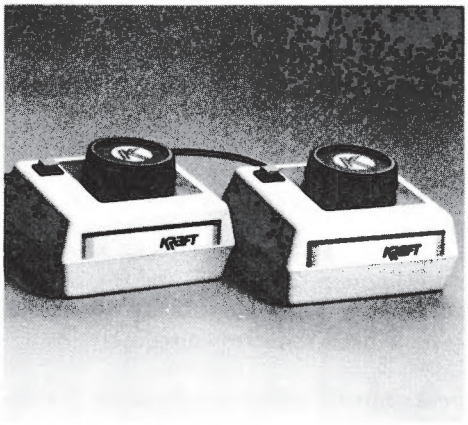


Fig. 9.4a Kraft game paddles.

Fig. 9.4b Kraft premium joystick.

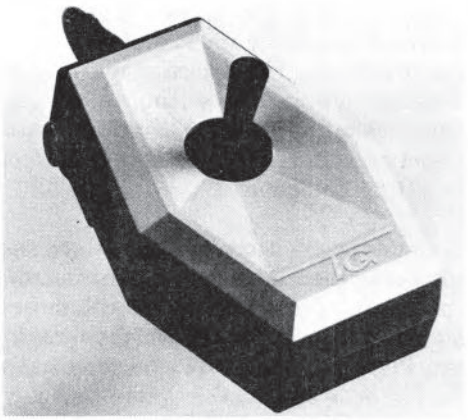


Fig. 9.4c TG Enjoystick.

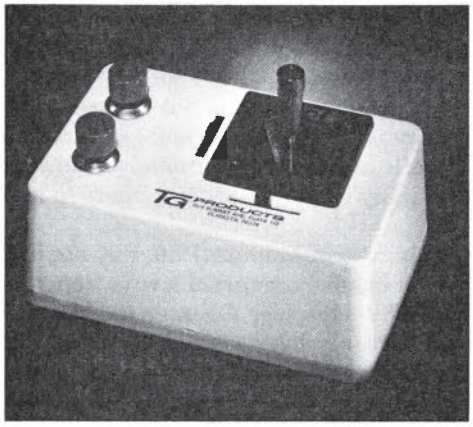


Fig. 9.4d TG Joystick.

In a Joystick or a Trackball, a single control affects both the X and the Y position inputs so you get convenient bidirectional control. Most have a "spring centered" mode in which the control snaps back to a central home position on release. This requires X and Y adjustments since different games require slightly different home points. There is usually also a "free floating" mode in which the stick stays where you left it when released. When you choose a joystick, you should be sure it is easy to change modes and do the fine adjustment.

The Kraft Premium Joystick (see Figure 9.4) is based on a design from radio-control airplanes and other remote devices where you place your thumb on top of a fairly short stick with limited travel. This joystick is unique in that you can independently set either the X or the Y axis to free floating or spring centered mode rather than being forced to switch them both at the

same time. In addition, one push button is placed on top with the second one on the side so it is easy to use one with your thumb and one with your index finger, while the TG joystick has both buttons on top. The Mach III joystick from Hayes Products has a very different emphasis with a heavy hand grip on the stick and a button placed on the top of the stick.

Fig. 9.5 The TG trackball. The trackball is used for a distinctive feel in which there is a low control to movement ratio.

The TG trackball (see Figure 9.5) is used for a distinctive feel in which there is a low "control to movement" ratio. This means you can spin the thing hard to get a motion rather than adjusting a joystick a few eighths of an inch.

Game Port Extenders

Particularly if you own an Apple II/II+, you may find that more often than you'd like you have to open up the Apple and plug and replug different kinds of hand controls into the game port. There are several products which can take a lot of the bother out of this. The simplest is the Scooter 0-Force X-Port (zero force external port). You plug one end of the device into the game socket and mount the other end outside your Apple. The external socket has a release and clamp lever so you can put the game plug in place and then clamp it with much reduced risk of bending pins.

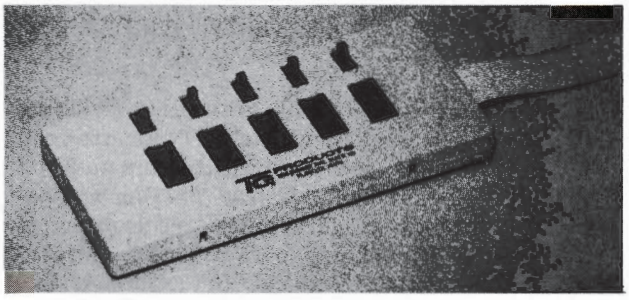


Fig. 9.6 Select-A-Port from TG has several different game port configurations, but also simplifies changing from joystick to paddle, etc.

The Mimco joystick has a full game socket in its side so you can plug your other controller into the Mimco device without any unplugging. The second device can be assigned to PDL2 and PDL3 or it can be switched in as the only active control.

The most elaborate game port upgrade is the Select-a-Port from TG products which has five ports (see Figure 9.6). Three of the ports are standard game ports and can be switched in one at a time. The fourth is configured to operate PDL2 and PDL3, and the fifth is wired specially to handle sensitive controllers which would be affected by sharing the input lines with other devices.

Graphics Digitizers

The main purpose of a graphics digitizer is to copy an existing image into digital computer memory. For many applications, a mouse, touch pad or light pen system will be much more convenient for free hand drawing. Another potential strong point for digitizers is that they can be arranged to encode information at a much higher level of detail than can be displayed on the screen.

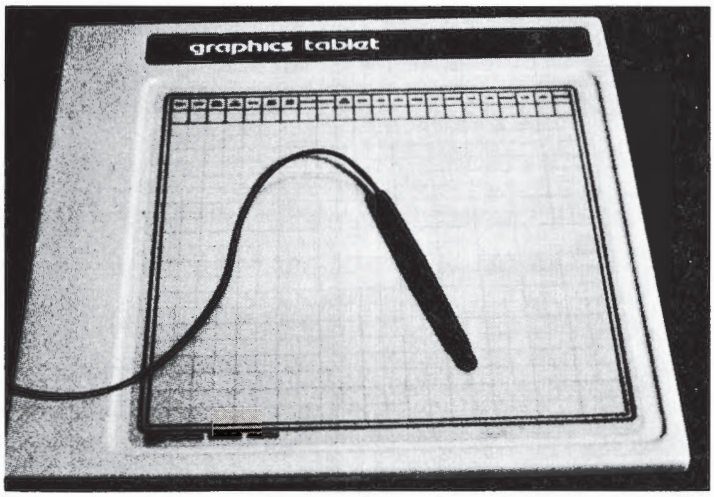


Fig. 9.7 Apple graphics tablet.

The digitizing tablets from Apple Computer and from Summagraphics are based on drawing boards in which a magnetically sensitive grid has been embedded. The grid can detect the presence of a special stylus through several sheets of paper. The Apple Graphics Tablet (see Figure 9.7) has a minimum detection spacing of .039 inches which is actually a lower resolution than the Apple II Mouse.

A simpler and much less expensive approach to designing the electronics is to build a digitizing arm with a game paddle at each joint. The Apple's built-in game paddle A/D converter can detect 256 different angles at each joint. The resolution you get for digitizing from an 8 1/2 by 11 sheet of paper is nearly as good as with the Apple Graphics Tablet. The Versawriter from Versa Computing (see Figure 9.8) has a two joint digitizing arm based on this principal and the Space Tablet from Micro Control Systems has three joints for digitizing real three-dimensional objects.

Although these systems do well with regard to resolution and speed of digitizing, there are problems with "linearity." This means that the space between steps is not completely equal across the full range. In addition, there can be distortion of one axis relative to another if the paddle potentiometers at each joint are not calibrated to operate with identical sensitivity.

Video Digitizers

If very high resolution is not important, then the fastest way to digitize a complex pattern is with a video camera interface. Once the image is entered, you can always come along later and edit the image with a light pen or high resolution digitizing pad.

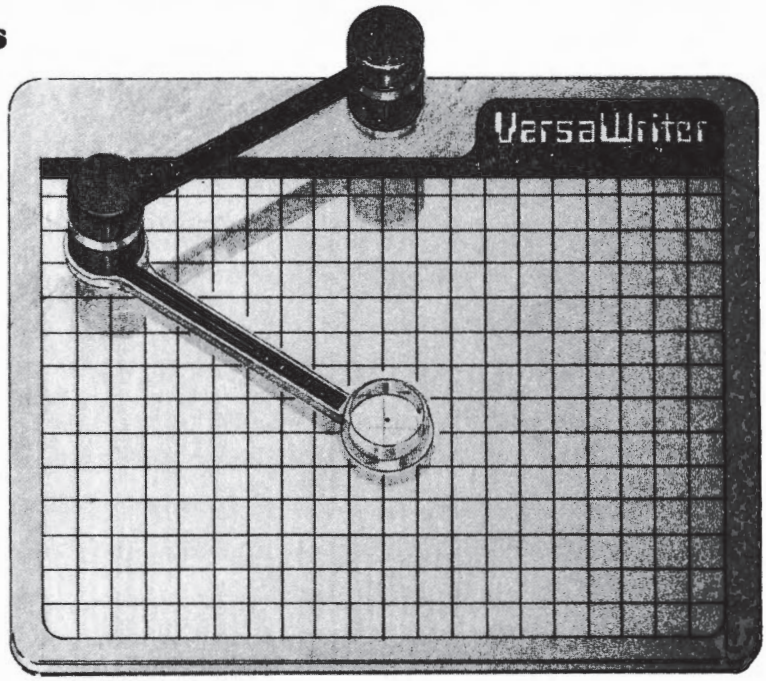


Fig. 9.8 Versa digitizing tablet.

The output of a video camera is an analog waveform with higher voltages for bright spots and lower voltages for dark areas. The output traces a raster pattern (see Chapter 5, Figure 5.1) which scans through the entire image 60 times a second. All that is needed is an appropriate high speed A/D converter (see Chapter 14). To get a good description of the image you need to know the "gray level" of each dot. An eight bit A/D converter would give you 256 levels of gray at each dot.

In reality, fast, high resolution digitizing of video output is an extremely challenging technical problem. To do real time digitization at the level of resolution of the Apple high res screen, you have only about 140 nanoseconds (billionths of a second) for each pixel. There are Flash A/D Converters which can do this fairly easily, but to get eight bit precision (256 gray levels) you have have to spend well over a thousand dollars just for the digitizing chip. Fortunately, the 64 gray levels you can get from a six bit flash converter is fairly adequate and these chips cost only $300 or $400. This is the basis of the video digitizer from Micrographic Images Corp. which they use with their ultra high resolution graphics system (see Chapter 7).

For most folks interested in video digitizing, there is no need for such expensive electronic heroics. As long as you are digitizing a fixed image, it is no problem to just let the image sit there and slowly digitize it over a period of a full second or so. This is also much more reasonable in light of the speed and storage capacity of the Apple computer since a fast machine language program can only accept a new byte about once every 30 or 40 microseconds (millionths of a second) and since there is only room for a small number of frames in main RAM.

The Dithertizer //e from Computer Stations (see Figure 9.9) uses an efficient and inexpensive digitizing system based on what amounts to a one bit A/D converter. It actually does real time digitization, but it collects one gray level at a time. It digitizes an image with two gray levels in a single full scan of the camera. Then the intensity setting of the converter is incremented a little, and the image is digitized again at another level of brightness. After 64 full passes of the raster, you have your 64 gray level image and just over one full second has passed. The mathematical process of overlaying the multiple brightness images is called "dithering."

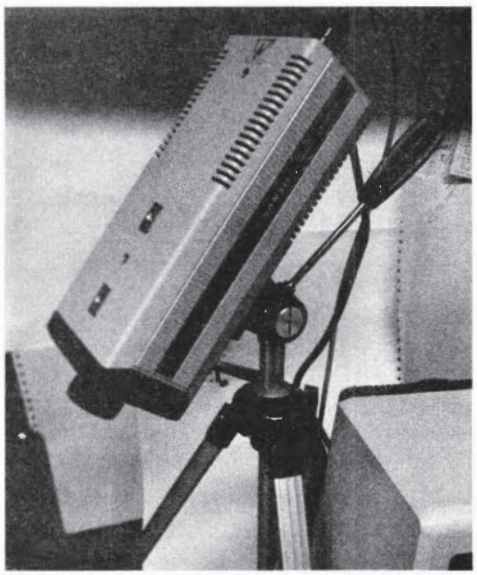
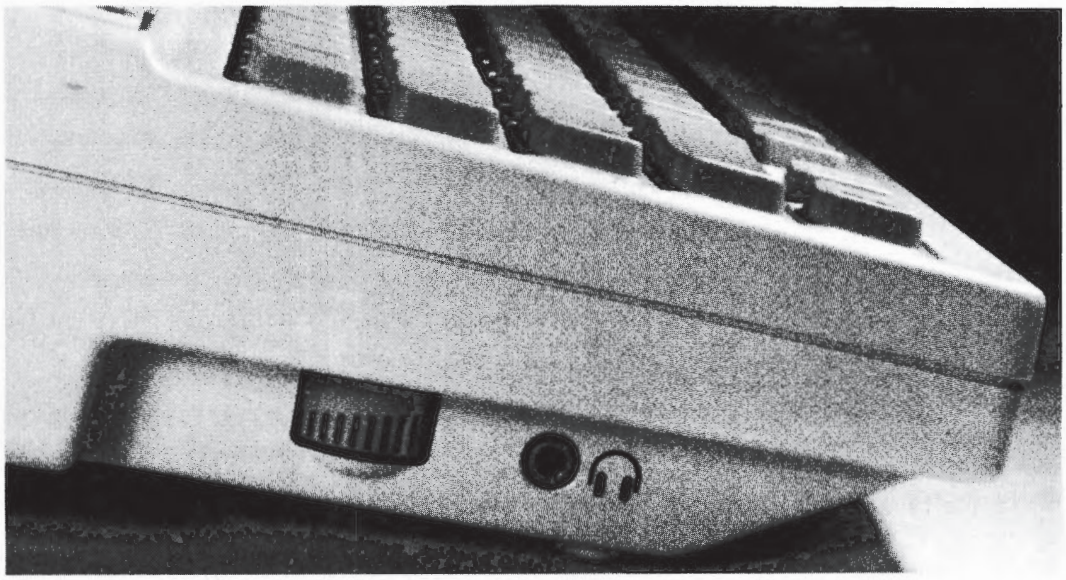


Fig. 9.9 Computer Stations supplies this video camera which has the proper synch signals to work with their video digitizing system.

The Dithertizer //e system lets you use arrow keys or the game paddles to adjust brightness and contrast, and it can be operated with a II+ if you get an extra adapter. The Dithertizer //e software has been written to let you use //e super high res (560 by 192 dots) if you have a revision B //e (see Appendix D) and at least 64K of extra RAM in the auxiliary slot. The primary weakness of the system is that it requires an industrial type video camera with an external synch signal rather than an NTSC type of video camera such as are used in home video systems. Fortunately, these industrial type cameras cost only a few hundred dollars.

There is a slower system called the DS-65 Digisector made by Microworks which has the advantage of being compatible with either an NTSC or an industrial type video camera. The Digisector uses a different approach to digitizing in which you select the coordinates to be digitized one by one. As each position is identified, it is fully digitized to six bit precision (64 gray levels) but this is a software intensive process that proceeds at a rate of one pixel every 126 microseconds. This means that it takes about eight seconds to fully digitize a single image with 256 by 256 dot resolution.

This full digitizing time is required if you need to do the whole screen; however, if you need to follow only a small area, and write your own data collection routine, you can move the "digitizing cursor" around and collect data fairly rapidly. This is what Microworks calls "random access digitizing." To help you do this, the Digisector board provides a real time analog video output with the position of the digitizing cursor marked so you can watch what you're doing on a separate CRT monitor and even write a program which lets you control the area of digitizing with game paddles.



Chapter 10

Sound and Music

We can hear anything that will make our middle ears vibrate. Most of the time this pertains to those airborne vibrations which we call sound. Hearing, per se, is due to a spectacularly precise analysis of incoming sound which is performed by our inner ears. That analysis describes any sound in terms of two component qualities: frequency and amplitude.

In fact, there is a remarkable richness of detail in the frequency and amplitude composition of such standard everyday sounds as clicks and crashes, musical notes, and spoken words. Our inner ears sort all this out almost instantaneously and without the slightest conscious effort on our part. Unfortunately, one result of the ease of all of this is that we remain singularly bored by and uninterested in simple uncomplicated sounds. As a consequence, the production of sounds which we find interesting, pleasant or communicative is no simple task for a microcomputer.

Actually, aside from the gross storage inefficiencies of trying to use RAM or floppy disks as a recording media, a microcomputer can capture and reproduce sounds with wonderful fidelity. However, we want our microcomputers to listen to us as easily as they can accept instructions from the keyboard and to talk to us as readily as they put words on a video screen. Record and playback we can do with tape recorders. Therefore, much of the excitement and effort has gone into designing devices which permit a microcomputer to actively and dynamically assemble complex sounds, in real time, beginning only with their raw, underlying simple frequency and amplitude components.

Elements of Sound

Most of the elemental basis of sound is not intuitively obvious from our experience in the natural world. It's really fairly simple stuff, but it's just not what you'd expect.

For instance, the most central building block of all sounds is what we call the sine wave (see Figure 10.1). We've all seen pictures of these and read about them in math and physics courses, but it's difficult to explain just exactly why this particular structure is so pervasive in our natural environment. This is our standard way of representing pure vibration at a steady rate. It is more fundamental than the atom as a basis of matter and energy in the universe, and you're just going to have to get used to it for the duration of this chapter.

A pure sine wave can be described by an amplitude and a frequency (as shown in Figure 10.1). One of these waves can be used to drive a speaker, and if the frequency is between 30 cycles per second (30 Hertz or Hz) and 18,000 cycles per second (18 kiloHertz or kHz) the wave is

reproduced as a mechanical oscillation in your middle ear and detected as a sound by your inner ear. The 30 Hz signal produces a steady, very low tone, while the 18 kHz signal produces an ear splitting whistle. If the amplitude of the original wave is used to determine how much energy is being used to drive the speaker, then we will perceive the amplitude as loudness.

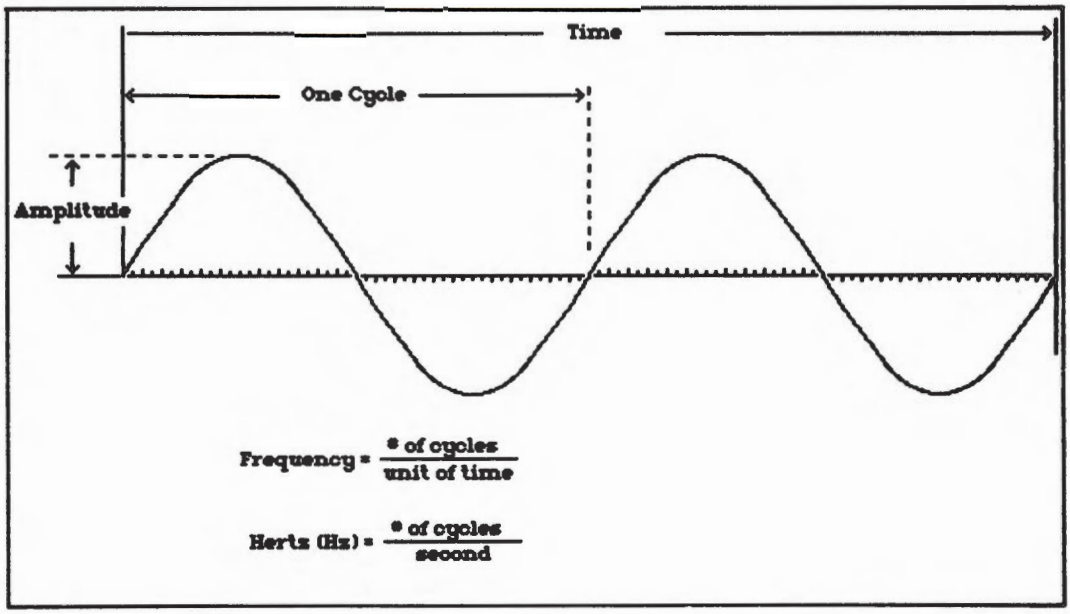


Fig. 10.1 A sine wave.

So far, we have been talking about what are called "pure tones." A pure tone is based on a simple sine wave with no adornment. Most of the sounds we are accustomed to hearing are actually composed of a large number of sine waves, each at a different frequency, and all mixed together at their source. To recognize a complex sound, the inner ear performs an analysis which breaks the sound down into its component frequencies. Incredible as it may seem, it is even able to determine the relative amplitude of each of the components. It is actually this array of frequency/amplitude pairs that the inner ear reports to the brain for further analysis and recognition.

It is also possible to construct electronic devices to do this kind of frequency analysis, or even to record the sound digitally and do the analysis mathematically. As a result of this ability, it has been possible to examine the frequency components of musical notes, spoken words, and other interesting patterns of oscillation. The results of this kind of analysis reveal more counterintuitive wonders about the composition of sound. It is these results, however, which provide the basis for music synthesis, speech synthesis and voice recognition.

Harmonics

Although most sounds are complex mixtures of frequency components, there is usually some regularity in the system which in effect permits all of the components to live together. Analysis of a sound usually reveals that it is composed of a single, dominant, "fundamental frequency" and a series of "harmonics." A harmonic is a sine wave with a frequency which is an integral multiple of the fundamental frequency, i.e., 1 x, 2 x, 3 x, etc. (see Figure 10.2).

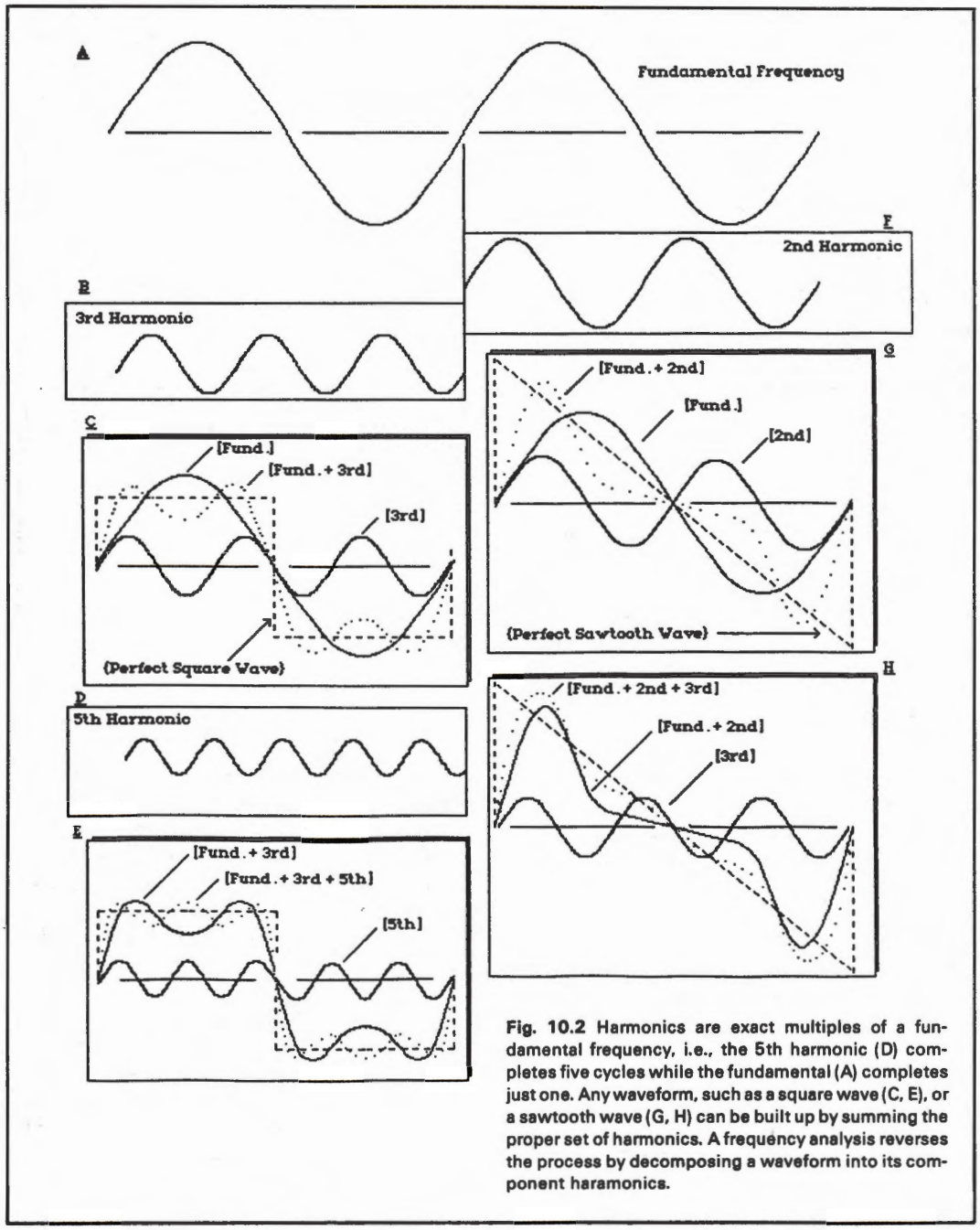


Fig. 10.2 Harmonics are exact multiples of a fundamental frequency, i.e., the 5th harmonic (D) completes five cycles while the fundamental (A) completes just one. Any waveform, such as a square wave (C, E), or a sawtooth wave (G, H) can be built up by summing the proper set of harmonics. A frequency analysis reverses the process by decomposing a waveform into its component haramonics.

A very striking example of how all this works is provided by the results of analysing a "square wave" (see Figure 10.2c and 10.2e). The fundamental frequency is the same as the frequency of the square wave, but the shape is very different. You can construct the shape of the square wave by adding in all of its "odd harmonics," i.e., 3 x, 5 x, 7 x, etc. As you can see, you need very high frequency harmonics to put a really sharp edge on the wave. By comparison, the results of adding both odd and even harmonics is shown in Figure 10.2g and 10.2h.

We often describe a wave such as the square wave or sawtooth wave in terms of the shape it assumes with all of its harmonics added in. A sustained musical note from, for instance, a trombone has its own unique shape or "waveform." One way you can simulate that sound electronically is to first analyse it to determine all of its frequency components. Next you would generate each of the components as a distinct pure sine wave and then sum them. If your analysis was good enough, you would produce a sound which was indistinguishable from the original. Stored in the computer, you would have just a series of eight or nine frequency/ amplitude pairs, but that would be a mathematical representation of the sound of the trombone.

In fact, you wouldn't have to store all of the actual frequencies. You could get by with a numerical value for the fundamental frequency and represent the harmonics by their number, i.e., 3rd harmonic, 8th harmonic, etc. This is very important. When you change the pitch of the trombone, you change the fundamental frequency, but you don't necessarily change the shape of the waveform. The frequency of, say, the third harmonic is now three time the new fundamental frequency; thus all of the components change their frequencies in unison and the overall shape of the waveform is maintained when pitch changes.

In summary, most sounds can be represented by a characteristic waveform (see Figure 10.3) That waveform can also be described as a "spectrum" of harmonic multiples of the fundamental frequency. To reproduce a sound you can either carefully trace its characteristic waveform or reconstruct it dynamically based on information on its component frequency spectrum.

Sound Histories

To fully engineer a sound, you have to do a little bit more than just get the waveform right. What we've put together so far is a complex waveform which is repeating at the rate of its fundamental frequency and which a human will recognize as the instrumental voice of a trombone; but at this point it's just a steady continuous tone. However, for many sounds it is extremely important to pay close attention to the first few instants during which the sound rises initially from complete quiet to its steady state. This initial rise is described by the musical term "attack."

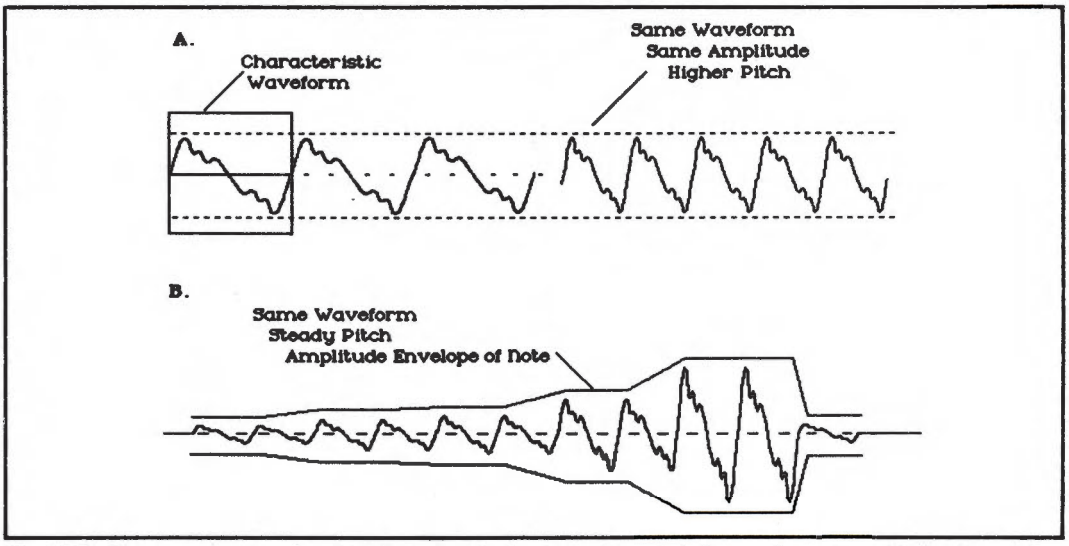


Fig. 10.3a The recognizable voice of an instrument is due to its characteristic waveform. When the fundamental frequency changes pitch, the harmonics change with it so the shape is maintained and we still recognize it.
Fig. 10.3b Attack, sustain, and decay, make up the amplitude history of a note. The envelope describes that history.

During the attack, the total amplitude increases at some steady rate until it reaches the amplitude at which it will be "sustained." Later, when the sound is ended, it will collapse through a characteristic "decay." These three events, attack, sustain and decay, mark out a "volume envelope" for the sound (see Figure 10.3) and are also called the sound's "amplitude history."

For truly accurate modeling, you also have to allow for changes in the shape of the waveform itself in different parts of the envelope. The mature waveform may not fully establish itself in the early phases of the attack, and it may change again during decay. In fact, it is often the changes during attack and decay which are the most recognizable aspects of a sound. This is particularly important when you are considering the rapid sequencing of the sounds that make up human speech. These shifts in the frequency spectrum of the sound are its "frequency history." The "robot-like" quality of some speech synthesizers is due in part to ignoring these changes during attack and decay.

Waveform Quality

It is therefore possible to synthetically create instrument sounds and human voices which are of dramatically good quality. However, to do this you need a very complex and elaborate electronic system, based on quite sophisticated knowledge of the sound you are modeling. To do this well is expensive, and is not always necessary.

The simplest and most economical electronic sound systems generate simple square waves rather than sine waves or carefully shaped synthetic waveforms. As noted earlier, a square wave has a complex frequency composition from the point of view of a human ear; however, it is the natural, simple, on/off output of a digital system. You hear the high frequency components as reedy or tinny overtones, but the overall effect is a passable simulation of a pure sine wave tone. The benefit of this kind of sound generation is that the electronics and computation are simple and comparatively inexpensive.

One very different kind of sound which is worth mentioning is called "noise." This term refers to a sound which is composed of a random mix of frequencies which have no particular relation to each other. You can make a low frequency noise by throwing in a mix of lower frequencies, or a high frequency noise, or "white noise" which is supposed to be a random mix drawn from the full range of audible frequencies. The frequency spectrum of white noise is "flat" because none of the frequencies has more amplitude than any other.

Whether your source of sound is a carefully modeled waveform, a raw square wave or noise, you can still control pitch, volume, and also the shape of the amplitude envelope. Sound effect systems and simple music systems can be based on square wave and noise generation. You only need modeled waveforms if you want to reproduce the recognizable "voice" of a selected real musical instrument. To synthesize recognizable human voice, however, waveform shape is most of the ball game, so square wave based systems are pretty much out of the question.

Sound Effect Generation with the AY-3- 8910

The least ambitious task in computer based sound generation is to make the tones, beeps, crashes, and buzzes that do wonders in spicing up a video game. One neat little package that will do a lot of the work for you is the General Instruments AY-3-8910 Programmable Sound

Generator (PSG). This chip turns up along with other features on the Waldo board from Artra (see Chapter 11), the Arcade Board from Third Millenium Engineering, the Sprite II and Supersprite from Synetix (see Chapter 7), the Mockingboard Sound II and Mockingboard Speech/Sound from Sweet Micro Systems and even in the sound effects section of the new Personal Speech System from Votrax.

The AY-3-8910 can't really be used for speech generation or for anything recognizable as instrumental music, but it is very versatile and also provides a good opportunity to step through some of the basic elements of a sound generation system. It is designed to accept a few descriptive numbers from a microcomputer and use this information to go about sculpting and producing the desired sounds.

The chip is based on three square wave generators and a noise generator. The three square wave generators are treated as completely independent channels and there is no provision in the chip for summing their outputs.

The programmer selects the frequency of the square wave by giving the AY-3-8910 a 12 bit number (1 to 4,095). The chip picks up the one megahertz signal from the Apple's built-in clock and divides it by 16 times this 12 bit number to produce a range of frequencies from 1 Meg/(16 × 1) = 75 kHz to 1 Meg/(16 × 4095) = 15 Hz, thus spanning the human audible range. So, although you have no control over the shape of the waveform which is the chip's "voice," you do have fairly good control over pitch.

Now that you've selected a frequency for the square wave, you can go about shaping an amplitude history envelope for it (see Figure 10.4). The AY-3-8910 offers two modes for controlling the envelope. One is direct control from the microcomputer. In direct control mode, you could let the user control the tone by pushing and holding a key on the keyboard. When you detect a keypress, you start feeding a series of steadily increasing attack amplitudes to the AY-3-8910, you maintain maximum amplitude while the key is held down (using the //e Any Key Down feature; see Chapter 8), and then step through a steadily decreasing decay when the key is released.

The alternative is an automatic mode in which you select from among several simple built-in envelope profiles and then let the chip do all the work. The AY-3-8910 will also accept an instruction to step through the envelope again and again at a specified frequency (see Figure 10.4) much like a string of quarter notes played on a kazoo or as a rapid fire feature on an audible raster blaster game.

Whichever mode you are using for shaping the amplitude envelope, you are allowed just 16 different settings (a four bit number). The actual output from the system is generated by a Digital to Analog Converter (DAC; see Chapter 14) which produces an output current proportional to the desired amplitude. The output for each channel is based on the state of the square wave frequency input and the four bit number currently being sent from the envelope generator and/or amplitude control.

The actual output from the DAC is logarithmically related to the amplitude value. This is because the amplitude sensitivity of the human ear is based on a logarithmic scale. A human ear listening to this output would perceive it as if it were regular steps.

The hardware specification sheet from General Instruments provides instructions on how to simulate gunshots, explosions, "laser sounds," whistling bombs, wolf whistles, and race car sounds. All of which gives you a fairly clear idea of the intended market for this device. This

chip is used in seven different Apple products from five manufacturers, so it's possible that it will begin to develop some sort of following among commercial game programmers. If you buy one of these products though, you'll have to be sure you're given documentation on how to program it or ready-to-run software which uses it to advantage.

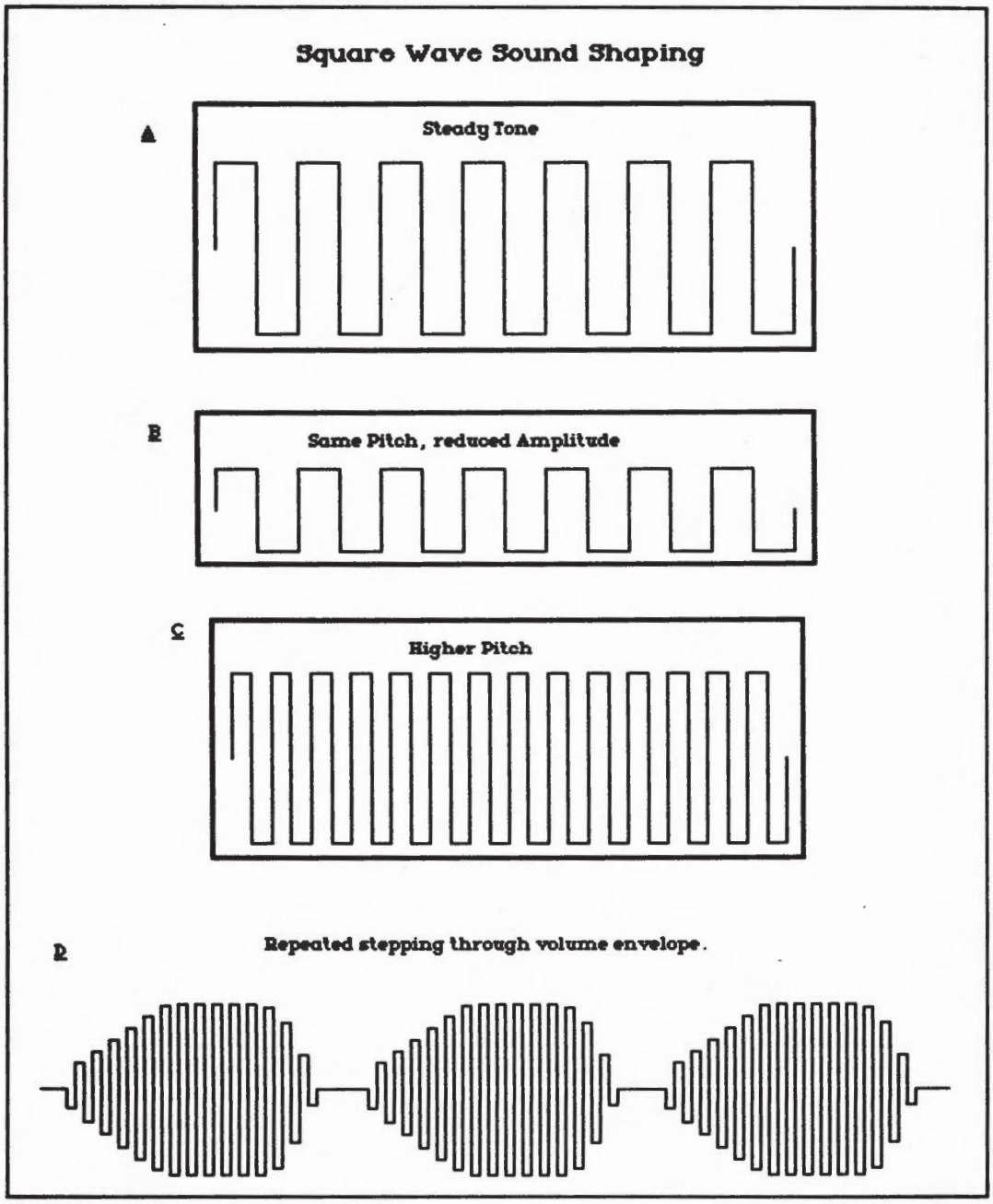


Fig. 10.4 Simple sound systems let you control pitch, amplitude, and amplitude envelope, while offering only a single square wave voice.

Square Wave Music

A sustained square wave tone sounds a little bit like a sound from an inexpensive electric organ; so if you've got control over pitch and note duration, you are effectively in a position to play out music with a square wave sound generator. The most important need is to bridge the gap between the standard musical notation with which musicians are familiar and the numerical orientation of the hardware. This is a software problem which was first solved successfully for the Apple by ALF Products in 1977 when they released their first Entry program for their MC16 Music Card.

While you are writing or entering music, the Apple's video screen is set up as a graphic representation of a musical score. The musician writes music by positioning standard notes on the staff. The fast machine language subroutines hidden deep within the software package can then read the score and issue appropriate commands to the hardware to play the song as written.

The MC16 has three "voices," each of which can be set to its own frequency. This lets you simulate the effect of playing an organ with three fingers down at a time, or of three different organs, each being played with one finger by a different person. The ALF MC1 sells for the same price but has nine voices. The difference is that between the time the MC16 was designed and the design of the MC1, Texas Instruments produced a sound generator chip which could pack all of the features into a smaller, less expensive package. You do have better control of frequency and amplitude with the MC16, but the MC1 has been more popular because of its nine voices.

When composing for the MC1, you write nine different lines of music, one for each hardware voice. The user interface is based on paddles and pushbuttons. You use one paddle to pick what kind of note you want (whole note, 64th note, etc.) and you use the other paddle to position the note on the musical staff, thus assigning it pitch. You can work in a four and a half octave range and you can hear a note when you enter it. There is also a provision for writing in percussive sounds by changing three of the lines of music from pitch to white noise mode. The music you write can be saved to disk and can be printed out on a dot matrix printer.

There is a very similar product from Applied Engineering called the Super Music Synthesizer which has 12 voices, four of which can be switched into percussive mode. There are actually "albums" of music for the ALF MC1 and you can play ALF albums on the Super Music Synthesizer. As with the MC1, you can shape the envelope of the notes, but an added feature is the ability to control the speed at which the song is being played by turning the knob on a paddle.

Instrumental Music Synthesis

The most important music hardware for the Apple II is the Music System from Mountain Computer. This hardware differs from the sound generators discussed earlier in that it provides the means for detailed crafting of the shape of the waveform. You can produce performance quality renditions of hundreds of different instruments as well as creating unique new instrument sounds.

The software from Mountain Computer is similar in spirit to the ALF software; you write music by placing notes on a musical staff using paddles and a light pen. However, the Music System is used as the basis for the Alpha Syntauri (see Figure 10.5) and the Passport Designs

keyboard synthesizers. Thus, with one of the keyboard systems attached, you can play as if with a piano, but the notes are played out in real time and automatically written into the score by the computer. This is therefore a true computer based musical composition system and it has achieved widespread acceptance both among professional composers and performers and among teachers and students of music theory.

Virtuoso Hardware Makes the Music

The input systems and software from Mountain, Passport, and particularly from Syntauri, are all so well done, "user friendly," and graphically sophisticated that virtually no user of these systems needs to know much about the detailed workings of the hardware. This is almost a shame, because the Music System boards from Mountain Computer represent the most daring and brilliantly executed electronic design of any Apple II peripheral. Those MIT types who remain unimpressed by mere music may just find the Music System aesthetically pleasing simply from the point of view of microcomputer electronics design.

The starting point in the design is the need to generate analog signals with frequency components near the high end of the human audible range, which, for the Music System, comes to 15,625 Hz. A Digital to Analog Converter (DAC; see Chapter 14) must get new samples at about twice this rate to get reasonably good reproduction, so you will need to send update information to the DAC at a rate of 31,250 Hz or once every 32 microseconds. To make things short and sweet, the Music System uses Direct Memory Access (DMA; see Chapter 27) to grab data from memory and shove it into the synthesizer chips without the involvement of the 6502.

At this rate of doing DMA once every 32 microseconds, the 6502 gets to step through 32 of its normal one microsecond clock periods, freeze briefly while the Music System does DMA, and then go back to work. If the Music System had two separate music channels, each with its own DAC, then it would have to do DMA twice every 32 microseconds. With four channels, you'd have to stop for DMA every eight microseconds. Eventually, as you kept adding channels, you'd be doing DMA so fast that the 6502 would never get any time for itself to get processing done. The practical limit is to do DMA every two microseconds, which means you can feed data to 16 separate music channels. Which is what the Music System does.

This effectively means that the Music System runs completely interleaved with the 6502, as if each were on a 500 kHz clock. The rate of data flow for the Music System is 500K bytes per second, which is much faster than what the 6502 actually does, and is also striking in that it is 20 times faster than what is achieved by the high performance scientific data acquisition systems designed for the Apple (see Chapter 15).

DMA Access to Waveform Tables

To understand how the DMA is supervised, and the nature of the data that is being moved, you need to flip back to the discussion of waveform composition. The Music System is able to simulate specific musical instruments because it can produce copies of their natural waveform shapes. The construction of these waveforms from their harmonic components is described shortly, but for now, just consider that the objective of the system is to step the DAC on through a series of samples of the waveform.

The shape of a waveform is stored as 256 samples. Each sample is an eight bit number which represents one of 256 slices through the waveform. This information doesn't describe the frequency at which the waveform will be played, only its shape. The 256 bytes which describe the waveform are stored in a "Waveform Table" in the Apple's main RAM.

When a Waveform Table is stored, its first byte is always placed on a "page boundary" so that all 256 bytes in the Table will be on the same page. Each RAM location in the Apple can be described by a page number and an offset into the page (see Chapter 21). Thus, by storing the Waveform Tables in this manner, the Music System assures that all of the bytes in the Waveform Table have the same page number. Therefore, for the DMA system to step its way through a Waveform Table, all you have to do is tell it which page number to look at, and then let it cycle its way on through the 256 offset numbers in the page.

Pitch Control

You will recall that the Waveform Table doesn't specify the frequency at which the waveform will be played. The frequency is controlled by the system software depending on the pitches called for in the musical score. Once a pitch is requested, a frequency instruction is sent to the music generation hardware. This is the same hardware which supervises the DMA and which reads in the bytes of the Waveform Table. What the hardware does with the frequency command is set the rate at which it will read through the Waveform Table.

Every 32 microseconds a music generation channel initiates DMA and siezes control of the Apple's address and data bus. While it has control, it announces the page number of the Waveform Table it is reading from, and then it must present an offset into that page in order to generate a complete memory address. At slow frequencies, the offset number will be the same through many rounds of DMA, advancing only very slowly. At very high pitches, the offset will advance with every DMA.

The Time Base for Playing Out the Envelope

The amplitude of the output signal will be based on information about the envelope at a given instant in time as well as on the value of the waveform sample at that instant. The music generation hardware will continue to cycle through the waveform at an assigned frequency without any further attention from the 6502, but the amplitude information in the envelope must be updated constantly so that notes can begin, rise in intensity, sustain for their assigned duration and then fade. This sequence is a timed series of events dictated by the notes on the score which the Music System is reading from.

In order to get accurate control of the time base for reading through the music, the system needs an accurate timer which it can read regularly without confusion, no matter what other complex tasks it is carrying out. In the Music System, this is carried out by the use of "interrupts" (see Chapter 27) which are generated by the Music System hardware. A timer on one of the music boards generates an interrupt every eight milliseconds. When an interrupt occurs, a simple routine increments a counter in the Apple's memory space.

The timer which generates the interrupts is, in effect, a real world time device, so it proceeds with unerring regularity independent of DMA, processing, or even other interrupts. Thus, the 6502 can always read the value in the counter to find out exactly how far into the musical score it should be. The eight millisecond counter interrupts provide the time base for the software system and also represent the ultimate time resolution for making changes in frequency or amplitude. With every advance of the counter, the 6502 advances to the next slice of the musical score and sets up the values which will operate all of the music generation hardware for the following eight milliseconds until the next interrupt and counter advance.

Composer's Model of the Hardware

Although you don't have to know anything about the hardware to use the Syntauri or Passport system, most composers need an appropriate mental representation for the elements of the sound producing system. This "composer's model" attends to the most basic musical effects of the system rather than to the electronics.

In the composer's model, the Music System is made up of 16 programmable "oscillators," eight for the left speaker and eight for the right, each of which can generate musical sound. As with any instrument, you can program an oscillator's pitch and volume to produce a melody in the playing out of a musical score. The extra dimension in programming the oscillators of the Music System is that you can also program the "timbre" of an oscillator which means you can choose whether it will sound like a violin string or a like a trumpet.

Musical Score for Instruments

The first layer of programming is completely identical to writing out a musical score for a human musician. Each line of the score has information on register, pitch of individual notes, and the time sequence of the piece, i.e., quarter notes, triplets, etc. If you own only the Music System itself, you use the Apple's standard typewriter keyboard, the game paddles, and a light pen to write notes on the musical staff, but if you have the Syntauri or Passport versions you can write the score by hitting keys on a piano style keyboard (see Figure 10.5).

If you purchase any one of these three systems, it will come with software which will handle all of the deeper levels of programming for you. You can just select the "preset" instruments you would like to use and then go on about writing music much as you would write text with a word processor.

You can also write out different lines of music for the different instruments you assign to the various hardware oscillators. These can all play simultaneously, like a small orchestra, and this also permits you to use the Syntauri or Passport system as a multi-track tape recorder. You can key in a new line while listening to the previous lines. On playback, you can see any one of the lines written out in musical notation on the screen, so you can immediately change notes to alter the arrangement.

Thus, working just at the level of the supplied "preset" instruments, this is a very powerful device for a composer or musician. However, the true power and much of the educational value of the synthesizer comes from the ability to work at a slightly deeper level.

Modeling the Instrument Sounds

The first level down into the depths of the system is the shaping of the envelope of a note. This process is explained earlier, and is no more than a means of adjusting the quality of the attack, sustain and decay of the volume of a note as it is played by an instrument.

Most real instruments have natural properties of envelope shape which is part of what you recognize as the unique voice of that instrument. However, this is one of things that music teachers and students spend a great deal of their time working with in the development of technique. The Music System lets you make direct changes in the envelope which you can see represented on the video screen, and it permits you to hear the effects of the changes you experiment with.

Defining Instrument Waveforms

The really exciting thing about these synthesizer systems for composers oriented towards music theory is the ability to directly control the harmonic composition of the voice of an instrument. As outlined earlier, most of the musical sounds we are familiar with are recognizable because of the number of harmonic multiples of the fundamental frequency and because of the relative amplitudes of the various harmonics. When summed, the fundamental frequency and the particular set of harmonics result in a particular waveform which is characteristic of a particular instrument.

You can create an instrumental voice by selecting the harmonic components and adding them together with the Music System. This process of "additive synthesis" can be done either with the hardware or with the software of the system. Since you have an array of 16 oscillators,

you could conceivably set all of them to play pure sine waves, with each one assigned to a different harmonic and amplitude. The sounds would then be combined on their way to the amplifier. However, the assembly of the waveforms can also be done mathematically by the software.

The principal limit on the complexity of a waveform shaped by the Music System software is that there are just 256 time samples allowed in the waveform and the relative amplitude of different segments of the waveform cannot vary by more than a factor of 256. Many instruments can be defined in software to play out on a single oscillator, but by constructing two waveforms that play out aspects of the voice on two oscillators, you can usually achieve any desired effect.

Choosing a System

The manual which comes with the Music System is a remarkably detailed and well written document of nearly 100 pages. It reviews music and sound theory, explains the composer's model in great detail, and steps you through the use of the accompanying software. However, without the Syntauri or Passport enhancements, the unadorned Music System has some frustrating aspects. When you compose music on the screen, it must first be "compiled" into a play file before you can hear what you have written. There is a similar time lag between the design of a new waveform and when you can hear the result of the new shape.

Both Syntauri and Passport allow you to hear notes played out in real time as you press keys on their piano style keyboards and both offer a large number of well designed "preset" instrument voices. Further, Syntauri has a Quickwave package which lets you listen to many waveform changes in real time.

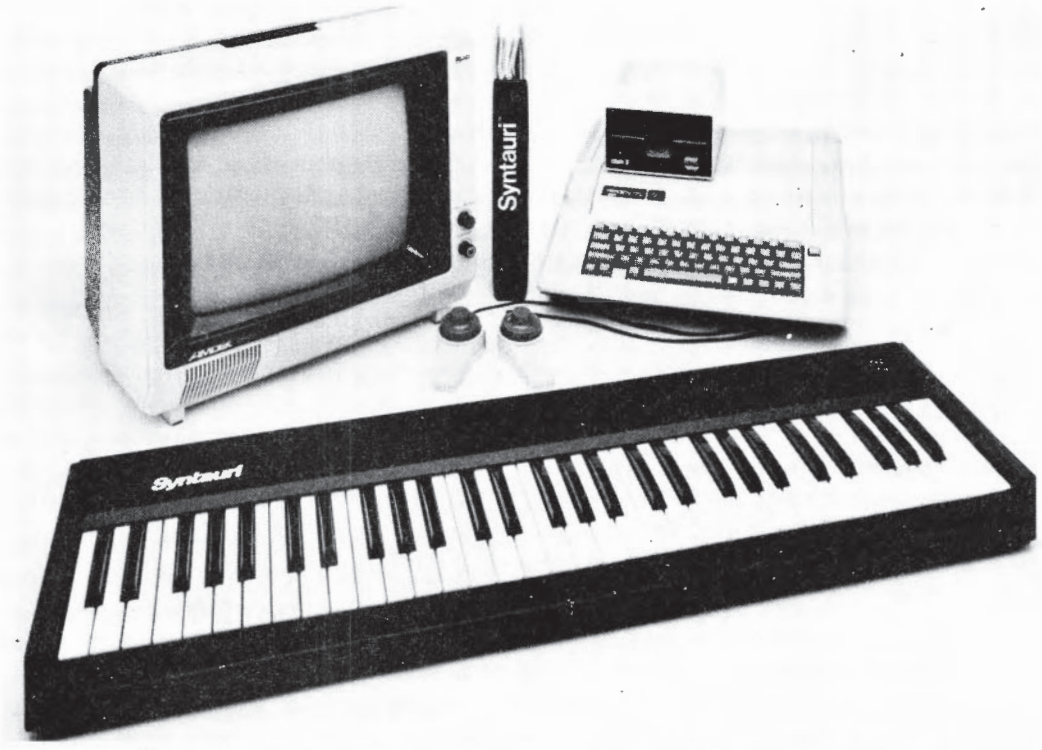


Fig. 10.5 The Syntauri synthesizer system includes the keyboard, software, and Mountain Computer instrumental music synthesizer electronics.

Syntauri has two different keyboards; a 49 note, four octave keyboard called the SMO4, and a 61 note, five octave keyboard called the SMO5 (see Figure 10.5). The SMO5 has "velocity sensitive" keys which gets you some tactile control over the loudness of notes as you play them. A normal piano keyboard is more sensitive to differences in force of impact rather than in velocity, but this does give you additional control. You can get two foot pedals for control of glide and sustain, and synchronization hardware to work with drum machines and tape recorders. The Soundchaser from Passport Designs has a 49 note, four octave keyboard.

These two companies compete with each other on the power and ease of use of their software, and both have a fairly strong line of software products. Many people will choose between the two 49 note systems on the basis of what a local dealer is familiar with or on particular software features. The SMO5 from Syntauri and their Hammond B-3 software with the ASO5C package is clearly the choice for the highest performance.

External Music Synthesizers

Those hardware afficianados who plowed through the section on the Mountain Music System boards will appreciate the remarkable microcomputer acrobatics involved in making those boards run. By attempting a challenging design, and sacrificing computational speed on the part of the Apple, the Mountain system with Syntauri or Passport enhancements achieves a remarkable level of direct interactive control over the music. A much simpler design approach from the point of view of the electronics is to build a complete synthesizer system external to the Apple in its own cabinet. The Apple is given responsibility for reading through the musical score and then sending information to the synthesizer over a communications link.

The Compumusic console from Roland Corporation looks like a mixer board with sliding levers. It has six preset instrumental voices which you cannot modify, as well as a percussion voice. You compose music in a special score system which has no particular resemblance to a traditional musical staff but which gives you the ability to use direct numerical descriptions of pitches and rhythms. The seventh and eighth channels on the console can be used to gain programmed control over separate analog synthesizers or drum machines, so you can use the Apple in place of a "voltage controlled sequencer" for a standard synthesizer.

With the action heating up in Apple music synthesis, Fender/Rogers/Rhodes has gotten into the act with their Rhodes Chroma music terminal and Apple interface. The idea of a "music terminal" is to view a digital synthesizer much in the same way as we are accustomed to thinking of a printer or video display terminal. The Rhodes Chroma is a complete, self-contained synthesizer with a five octave velocity sensitive keyboard, 16 oscillators, and program control over waveform shape as well as over time sequence and pitch. It has 50 preset instrumental voices built-in, all of which can be selected with switches on a control panel above the keyboard.

The Chroma can be used for performance in real time, and it has its own RAM memory. However, by providing an Apple interface, Fender/Rogers/Rhodes greatly enhances the versatility of the device. All of the commands which can be issued from switches on the control panel can also be sent from a stored music sequence in the Apple.

Music Terminals and MIDI

As more and more synthesizer manufacturers look to the route of interfacing with personal computers, there has been an anticipation of the need for a standardized communication code. This code would convey musical data just as ASCII code conveys text. Other terminal-like

devices such as printers and modems use ASCII based "escape sequences" (see Chapter 18) and command strings for communication, but this may not be adequate for music synthesizers. As discussed in the section on the Mountain Music System hardware, many of these systems require a very crisp rate of data flow.

The consensus has been to use a code called the Musical Instrument Digital Interface (MIDI). The mechanical specification calls for a five pin DIN jack, and the carrier system is based on an "asynchronous serial" (see Chapter 16) transmission similar to what is used for RS-232 modems and printers, but run at the comparatively high speed of 31.25 Kilobaud (31,250 bits per second).

Processing Digitally Recorded Sound

All of the systems described so far in this chapter are based on the synthetic production of sound relying on numerical representation of such properties as pitch, amplitude and harmonic composition. The alternative is to record real sounds and play them back, much as is done with the popular compact disk and laser systems. The difficulty with this is that a digitally recorded "sound image" has to contain a huge amount of data.

In the course of producing sound waves, the Mountain Music System synthesizer gobbles up data at 32,000 bytes per second for each of its 16 channels and so achieves full scale data flows of 500,000 bytes per second. Most of these bytes are generated from scratch, in real time, based on computations and descriptive parameters. If you wanted to play out a digital recording with the same amount of data, you would need half a megabyte of RAM for each second of sound. The full capacity of a huge mainframe with 16 megabytes of RAM would only supply sound for eight seconds.

Nonetheless, if you are willing to be a little less ambitious about the quality and complexity of the sound, and if you can make do with brief bursts, then digital recording and playback can be a viable option with some interesting properties. You can, for instance, pack a half second drum sound into a 16K ROM with full fidelity, but you may be able to get away with half the amount of sound detail and so get a full second into the ROM. If you can call the sound from ROM repetitively, then you've got microcomputer control of a real recorded sound.

The E-Mu Drumulator is such a ROM-based digital sound system. It is an external box which contains ROMs for 12 different kinds of drums and percussion instruments. It can be operated separately from the Apple, or you can connect it to the Apple with a standard RS-232C serial port (see Chapter 18) and use the Apple to sequence the drum sounds.

Another interesting application for digital sound processing is in the modification of real sounds to create altered sound effects. The DX-1 from Decillionix includes a card for an Apple II expansion slot which permits you to record and play back sounds, but which also provides for modeling the sounds while they are in memory. The record and playback features are based on an eight bit Analog to Digital Converter (see Chapter 14) and an eight bit Digital to Analog Converter.

Once you've captured a sound into memory, you can alter its pitch, add echo and reverb, or just generally fiddle around with a joystick control to see what kind of bizarre modifications you can produce. Once you are through modeling a sound you can then play it out and record it with a regular tape recorder or save it to disk for replay on the DX-1 system.

Chapter 11

Apple Speech and Hearing

Voice

The elements of human voice and speech can be understood in many of the same terms as were used in the overview of sound and of instrumental music. The human vocal tract acts like an orchestra with about 40 different instruments, each of which we call a phoneme. As with a musical instrument, we recognize a phoneme by the shape of its waveform. In the course of pronouncing words, we do something akin to operating a very fancy music synthesizer which can change its output from trumpet to violin to drum, shuffling among dozens of instruments at a rate of eight or 10 instruments per second.

Most phonemes, such as the "ee's" and "oo's" of vowels, are based on harmonicly rich sine wave sounds, while others such as the "ss" in fuss are based on modeling the hiss of white noise. To synthesize a human voice you need an appropriate controlled source of harmonic tones, a white noise generator, and a few other odds and ends to model the sound output. Just as with music, you must attend to the changes in frequency and amplitude during the attack, sustain and decay phases of a phoneme.

One additional concern with voice is that we vary the way we pronounce a phoneme depending on which phoneme we just finished with and which one we are about to start. You can produce recognizable synthetic speech with just 40 standardized phonemes, but to get good quality you have to include several variants of each which are called "allophones."

If you construct a system with a total of 128 available allophones and if you can see that you've got exactly correct timing for slipping from one to the next, throw in pauses and stops of appropriate duration, and make a few allowances for intonation. . .. Well, then you can synthesize voice which is astonishing in its natural quality and which is virtually indistinguishable from a tape recording of a real human. Wonderfully enough, there are Apple based systems priced at under $200 which can do all of this.

Phoneme Production by Humans

There are a variety of ways of classifying phonemes, but the most essential distinction is between "voiced" and "unvoiced." Voiced phonemes, such as the vowel sounds mentioned earlier, require the action of the vocal cords, while unvoiced phonemes just require air whooshing up from the lungs to blow around between teeth, tongue and lips. You can test this out by putting your finger on your Adam's apple and repeating the "oo" sound and the "ss" in fuss sound.

Vibrating Vocal Cords

You activate your vocal cords by just pulling them tight to partially obstruct the airflow out of your lungs, and the result is that they start to vibrate with a pitch dependent on how tight your throat muscles have pulled them. The vocal cords are actually embedded in flat sheets of tissue, all of which vibrate, but the loudest vibrations come from the cords themselves and for most people the fundamental frequency is between 150 and 250 cycles per second (men average 140 Hz, women average 230 Hz).

The frequency spectrum of the sound emitted directly from the vocal cord is just the raw material. It contains components ranging from 150 Hz up to 4000 Hz. Your frequency spectrum is part of what is characteristic of your own individual voice. Voice based security systems can use this spectral information to "fingerprint" you.

Resonating Chambers

As this complex waveform sets up air vibrations in your throat and mouth, it activates several resonant chambers scattered about. These chambers have certain favorite frequencies which they tune in on and use as the raw material for producing a fairly loud harmonic (see Chapter 10) of the fundamental frequencies. Each of these resonant chambers has its own favorite output frequency with some low pitched resonators emphasizing 200 Hz components, and others boosting frequencies as high as 2000 Hz.

With your vocal cords vibrating at a steady pitch and loudness you can produce different output sounds by mechanically reshaping the resonant chambers. When the shape of a resonant chamber is altered, its resonant frequency as well as its ability to produce a loud harmonic can be changed. You reshape your resonant chambers by positioning your cheeks, tongue, lips, etc., and you can even kick in a large extra resonator by dropping your uvula and hooking the whole thing into your nasal passage.

Formants and Fricatives

The ultimate output waveform for a phoneme is determined by the starting waveform from your vocal cords after it has been colored by several resonant chambers in your throat and mouth or even your nose. The sound from the vocal cords is called the "glottal pitch," the resonant frequencies from throat and mouth are called "formant frequencies," and your nose is politely termed a "nasal resonator."

The final bit of resonating and shaping done on some phonemes is carried out by the teeth, lips, and the tongue when pressed into contact with other structures. This kind of sound is called a "fricative" and describes the sound in the f, c, t, and v of the word fricative. The f is an unvoiced fricative and the v is a voiced fricative.

Electronic Voice Production

There are three major distinct approaches to electronic voice synthesis, all of which turn up in at least two popular products for the Apple. The simplest approach, "waveform encoding," pays no attention at all to the intricacies of voice frequency composition and operates more like a digital tape recorder with record and playback modes than like an actual synthesizer. You get good quality speech, but there's only room in memory for a small number of words.

The remaining two methods are both attempts to construct voice waveforms from their theoretical frequency components. The "Analog Formant" method uses electronic oscillators to emulate the vocal resonant chambers and what not. This method produces the notorious "robot-like" speech made famous by the Votrax Type 'N Talk used in the movie *War Games*. Voice quality is not great, but it is convenient to build "text to speech" synthesizers with unlimited vocabularies.

The most sophisticated speech synthesis methods generate an output waveform entirely through numerical calculations. One such method is called Linear Predictive Coding (LPC) for reasons which will be explained shortly, and it involves full scale mathematical simulation of the frequency effects of the different parts of the vocal tract. With the advent of single chip LPC devices built into products such as the Echo II from Street Electronics, it has become possible to synthesize very good quality voice with the unlimited vocabulary yielded by text to speech synthesis. A second computational approach is a startling programming tour de force named SAM (see below) which is available only for the 6502, and which permits mathematically based synthesis without any specialized hardware.

Waveform Encoding Methods

This discussion recalls the review of digital sound processing in the music chapter. There is no problem making a digital recording of remarkable fidelity; however, this requires a fairly huge rate of data flow. The situation is not quite as bad as for music, since voice is actually easier to encode than music in this type of system.

For a good recording you must sample the sound waveform at a rate which is twice as fast as the highest important frequency components. Most music has important components with frequencies over 15 kHz (thousands of cycles per second), so you need to take 30,000 samples per second, with each sample involving a byte of data. Thus music encoding accumulates data at a rate of 30K bytes per second; quickly overwhelming a 64K Apple. Most waveform components in the human voice, however, are below four kHz, so you can do very well with just 8,000 samples per second, and can get passable quality with 4,000 Hz sampling. At 4K bytes per second, you could easily load 10 or 15 words into the Apple's memory for storage to disk or for later playback.

Waveform ecoding gets you very good quality reproduction and you can even make out the identity of the original speaker, so there has been some effort to solve the memory capacity problem through some form of data compression. This is actually a lot less magical than it sounds and is also simple to carry out.

The most popular means of compressing the data is use a method called "delta slope modulation." When you are digitizing with a standard, eight bit Analog to Digital Converter (ADC; see Chapter 14) you make time slices through the incoming voice signal and within each slice you determine which of 256 possible levels of intensity is currently coming in. Thus you produce an eight bit byte of data for each sample. An alternative is to record the direction in which the waveform is changing rather than its absolute value.

In one popular delta modulation system each sample is a four bit number. The first bit tells whether the current sample is greater or lesser than the previous sample, and the remaining three bits are used to state eight possible angles of slope for the change. Thus you can fit two 4 bit samples into a byte of storage space, and you've doubled your storage capacity.

The Digitalker from National Semiconductor is such a delta modulated waveform encoding and decoding system built into a single chip. The Digitalker is used in the Micromouth from Micromint, and in the Cognivox system from Voicetek. There is a similar chip from Motorola called the LM3418 which is used in the Supertalker SD 200 from Mountain Computer.

The Supertalker lets you do your digitizing at a poor quality, low data flow rate of 512 Hz, and also at one kHz, two kHz, or a more reasonable four kHz. As with any waveform encoding system, the big problem is storage space and speed of retrieval from disk. You can hold a few seconds of speech in RAM, and then pack about five or six times as much on disk. It might have been nice to use this with a RAM disk, but Mountain chose to use a special disk format for the Supertalker, which means you're probably limited to working with a standard Apple drive.

Analog Formant Synthesis

When you are working purely within the digital realm of computers, sine waves are rendered as abstract mathematical constructions. However, the older "analog" electronics, on which our radios and stereos have been based, permits a much simpler rendering of sine waves. Looking ahead into the section on basic electronics (see Chapter 13), you will find that it is very easy to string together components such as resistors, capacitors and inductors in such a way as to produce finely tuned oscillating electric currents.

Since the objective in voice synthesis is to recreate the various vibrations and resonances in the human throat, mouth and nose, it might seem that tuned electronic resonators are appropriate parts for the job. This occurred to electrical engineers who built crude systems along these lines much earlier in this century.

The problem always was that to follow the complex flow of rapidly spoken phonemes in a word, you had to readjust the settings of all the resonating circuits about 100 times a second. This was not a reasonable task with levers and knobs. However, if you can collect together all of the necessary resonating circuits and effectively put a digital computer in control of the knobs and levers, then you've got a working system. This being the 1980s, you can now purchase a complete system with all the oscillators and the digital control circuitry neatly packaged into a small plastic DIP chip carrier for a few dollars. The neat execution of this little feat in their SC01 Speech Synthesizer chip has made the name Votrax the electrical engineer's equivalent of a household word.

Structure of an Analog Formant Speech Chip

There are thus two distinct segments which make up an "Analog Formant" speech synthesizer such as the Votrax chip. In one section you have all of the analog oscillators, and in the other section you have a small scale built-in microcomputer which takes care of all the adjustments and updating.

The organization of the analog section closely reflects our theoretical understanding of the human voice production system. There are two alternate sound sources, one called the "glottal pitch resonator," which plays the part of the vocal cords, and one called the "fricative resonator" which generates white noise. Most voiced phonemes need to get passed through at least three resonating chambers in the mouth and throat, and some need to resonate in the nose. Thus the chip is provided with three "formant resonators" and a "nasal resonator."

The digital circuitry is given direct control over the frequency setting of each of the six resonators, and it has volume control over the glottal and fricative source as well as over the

nasal resonator. These six resonant circuits and their nine control lines make up the analog section of the synthesizer.

During the course of attack, sustain, decay and transition, the exact settings for a particular phoneme change. Thus the digital circuitry views a phoneme or allophone as being made up from a series of "frames." Each frame is a list of settings for the nine control lines. Elegant analog formant synthesizers can step through 30 frames in the production of a single phoneme. In sequence, you could send the chip a one byte instruction calling for it to produce a selected phoneme. The built-in ROM on the chip would use this byte to pick out a set of 30 instruction frames needed to play out that phoneme. As each frame was executed, nine controls would be adjusted. Your single byte request would have caused the chip to issue 30 times nine equals 270 resonator instructions.

To call up six phonemes in a second of speech, you would need just six bytes. The chip would respond with six times 270 equals 1,620 resonator settings and a recognizable word would be spoken. You may recall that a second of good quality speech can require 8000 bytes of memory with waveform encoding methods, so the reduction to just six bytes by the analog formant method really does change the whole ball game.

Text to Speech Algorithms

Since the microcomputer only has to send a few bytes per second, it has gobs and gobs of time to mill around doing processing while it is waiting to request the next phoneme. One popular thing to have the machine do while it is waiting is to run through a special program called a "text to speech algorithm." This kind of program does exactly what its name suggests. It reads text from a standard text file and causes the synthesizer to speak it out. The principal task of a text to speech program is to translate from the alphabetic spellings of standard English into the phonetic spellings required for proper phoneme selection.

Some of the translations are quite straightforward, but when the algorithm encounters words like "know" or "Connecticut" or "ASCII" it requires some level of sophistication to get things right. This means that the quality of a text to speech algorithm depends on how many "rules" it uses for assignment and how many "exceptions" it is able to watch for.

Even given that the phoneme selection is done well, there are a couple of other factors which can cause trouble for the quality of speech. First of all, the use of just three formant resonators omits some degree of detail, and the initial source tones from the glottal resonator are missing many of the higher frequency components which we expect to hear in speech. Another limitation of the popular SC01 analog formant synthesizer from Votrax is that it has just 64 different "allophones" to draw on, while more elaborate systems may have twice that number.

The end result is what has come to be called "robot-like" speech. In fact, newer robots will use much better synthesis systems, sounding more like HAL in *2001: A Space Odyssey* than like the Whopper master computer in *War Games,* so this is probably better referred to as Votrax-like speech.

Nonetheless, the Votrax SC01 has had an important impact on the popularity of speech synthesis for microcomputers. It is used in the Speech II and Sound/Speech I from Mockingboard, the Sweet Talker from Micromint, Waldo from Artra, and in two products from Votrax, the Type 'N Talk and the Personal Speech System. The Mockingboard products and the Sweet Talker are inexpensive single board products, while Waldo is an elaborate multifunction system described in the section on speech recognition systems.

The Votrax Type 'N Talk (see Figure 11.1) is an external, terminal-like device which connects to the Apple via a standard RS-232C serial interface. You send characters to it just as you would send them to a printer or a modem and it speaks out the characters as it receives them. The text to speech algorithm work is carried out by a 6800 microprocessor in the Type 'N Talk cabinet. You can control the pitch of the speaker's voice as well as its volume only by adjusting knobs on the front, and you have no control over the speed at which words are spoken or of inflection.

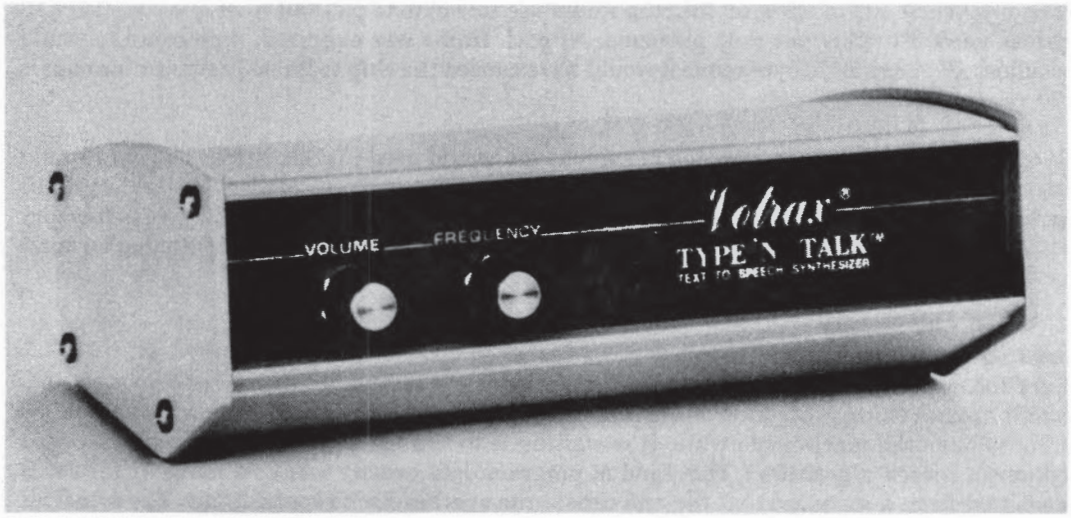


Fig. 11.1a Votrax Type 'N Talk acts as an external RS-232 speaking terminal.

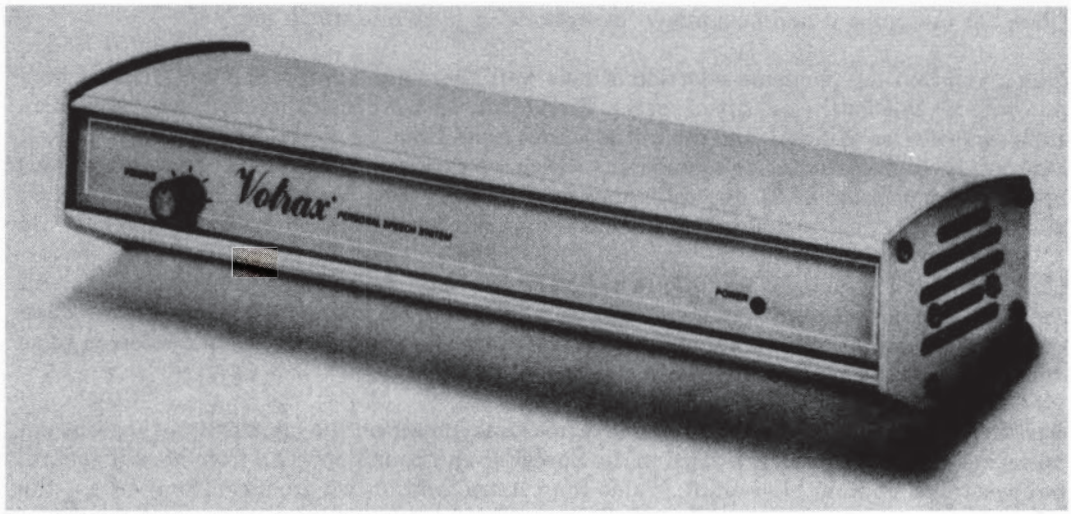


Fig. 11.1b Votrax Personal Speech System is similar in principal to the Type 'N Talk, but adds more sophisticated processing of the input and control of output quality.

Votrax has released a greatly enhanced product called the Personal Speech System (see Figure 11.1) which uses a Z-80 microprocessor to examine the incoming words for instructions on inflection, speech rate and volume. More importantly, the text to speech algorithm has been greatly enhanced. It knows how to handle a variety of abbreviations, will pronounce, i.e., "154" as "one hundred fifty-four," and even lets you load in your own exception tables. The system also includes a real time clock, an AY-3-8910 chip for tones and sound effects (see Chapter 10), and can interface with either a serial (//c) or a parallel port.

Computational Synthesis

As noted in the section on analog formant synthesis, it is simple to build electronic circuits to act as oscillators. Unfortunately, waveforms produced by that technique are so simple that they don't sound very much like natural human speech. With sufficient computational power, however, all of the waveform modeling can be done on a purely mathematical basis. The Software Automatic Mouth (SAM) from Don't Ask Computer Software works to replicate the phoneme waveforms using data from their fully formed frequency spectra.

A much more widely used commercial approach called LPC begins with waveform encoding and then performs a detailed analysis of the input to extract descriptive parameters. Later, when it is time to do synthesis, these parameters are used in what is an effective mathematical model of the vocal tract to recreate the recorded words. If you can get extremely powerful computation going, you can attend to all sorts of minor little details in the speech waveform which are essential to natural sounding voice.

This scale of computation far exceeds the abilities of most microprocessors. However, there is a two step approach to the math which makes it possible to construct simple, inexpensive voice synthesizers for machines like the Apple. All of the analysis of speech is done in advance on a large mainframe computer, and the spoken words are reduced to sets of parameters and coefficients to be plugged into equations at the time of synthesis. Then the synthesis is done by a special high speed math processor, designed specifically to carry out the equations of speech reconstruction.

When the large computer is used to analyze whole words, it will produce parameter lists for synthesizing those words. The parameter lists are sufficiently brief that there is space for over 200 encoded words on a single 16K ROM. However, it is also possible to extract the various encoded phonemes and allophones and store these rather than whole words. This means that the system can be used with a text to speech algorithm just as with the Votrax chip. The algorithm examines text files or incoming ASCII codes from a modem, and selects the necessary phonemes to pronounce the words. This gets you an unlimited vocabulary of natural sounding speech which is exactly the way you use the Echo II speech synthesizer from Street Electronics.

Properties of the LPC Coding Process

The equations that do the work accept an incoming digital waveform, and act like a filter to modify the shape of the waveform. Just as with the analog formant synthesizer, you begin with a glottal source and a noise source, but these sources output numbers instead of varying voltages. The equations which act as filters take the place of the various resonators in the throat, mouth and nose.

One additional advantage of starting with a digital waveform generator is that it is convenient to begin with a fairly complex waveform which is modeled after the harmonic complexity of an actual glottal pulse. The filter equations don't really care how complex the incoming waveform is, so you get a sort of free shot at improving the natural quality.

The actual calculations involved are not that spectacular: a couple of multiplications and a couple of additions. The problem is that you have to do all four operations about 100,000 times a second. The original process required four multiplications and two additions, but an enhancement made it possible to cut out two of the multiplication steps. That enhancement uses a standard mathematical "prediction" technique called linear regression and so the whole process has been given the name Linear Predictive Coding (LPC) speech synthesis.

Texas Instruments succeeded in building a specialized high speed math chip which does LPC calculations. This processor is packaged with a noise source, a glottal waveform source, a digital to analog converter and an audio amplifier in the TMS 5220. You use the 5220 in conjunction with a separate ROM chip or under direct command of the microcomputer.

To get it to produce a phoneme, you tell it the pitch, the amplitude, and 10 "filter coefficients" which affect the way it does its calculations. This set of 12 pieces of information is called an "LPC data frame" and, just as with analog formant synthesis, you need to feed it a series of 20 or 30 frames to produce a complete phoneme. The storage requirement comes to about 100 bytes per word. This is not as compact as with analog formant synthesis (6 to 10 bytes per word). More importantly, though, it is much better than waveform encoding which can require up to 8000 bytes per word to achieve similar quality.

LPC Synthesis Systems

The most outstanding voice synthesis system for the Apple is the Echo II from Street Electronics. The TMS 5220 chip it is based on turns up in several other products, including the SSB- Apple from Multitech, the Super Sprite from Synetix, the V101A from Vynet, and in Waldo from Artra. The two strengths of the Echo II are a wonderfully done software package and manual, and a text to speech algorithm.

The Echo II text to speech algorithm is not nearly as sophisticated as the new algorithm in the Votrax Personal Speech System, but it is the only one available for an Apple LPC product. If you experiment with phonetic spellings, you can produce speech a little better than the Votrax system will do automatically. The advantage of the Echo II is that you have the option of selecting from many hundreds of preassembled words. When you pull these in off the disk and put them into your BASIC programs, the system can produce a very natural sounding voice with full control over what is called "prosody"—pitch, rhythm, duration and intensity.

The system from Multitech is based on nearly identical hardware, but it is poorly documented, difficult to use, and offers no means of adding new words since you can't do text to speech. You are provided with a vocabulary of preassembled words, but many very important words are missing, and there's nothing you can do about it.

By adding a few additional kinds of hardware to their board, the folks at Vynet produced a very interesting device. Their LPC synthesizer shares space on the V101A with the circuitry usually used on modems for auto-answer and for touch tone recognition. The way this system works is that when someone calls on the telephone, the V101A will answer the phone and begin talking to the caller.

The interesting part is that it can tell the caller to specify what information they need by pressing a touch tone number. The V101A listens to the touch tone, and uses it like a selection from a menu in a standard piece of microcomputer software. It can respond to the request by speaking out a new list of touch tone selectable options or by reporting the desired information. The major liability of this system is that it comes with a limited, fixed vocabulary, and it's very expensive to have a special speech set made up. This may discourage individuals from checking this out, but it should be of substantial interest in businesses with a heavy burden of redundant phone answering.

Software Synthesis by SAM

When you consider all of the research and development effort that has gone into systems like LPC and the Votrax chip, you can't help being amazed by the accomplishment of the programmer who created SAM. There are no oscillating circuits, math processors or mainframes lurking in the background—the whole thing comes as a program on disk. The machine language code involved is fairly classy stuff, but then again it isn't something that no one else could have done. Thus the striking part is how simple and obvious it all sounds in retrospect in comparison with the Herculean quality of most of the other efforts.

All that's going on is a reconstruction of a phoneme waveform from its frequency spectrum. Knowing the harmonic composition of a phoneme and the rough shape of its envelope, the SAM software calculates the appropriate output waveform shape. This is all done in real time as allophones are encountered in the string of text. The equations generate about 7000 bytes per second, so the Digital to Analog Converter makes reasonably good reproductions of components of up to 3.5 kHz.

The equations also operate within the context of a set of rules which describe how an allophone should be adjusted to allow for its context in a particular word, and for easing the transition from one allophone to the next. The whole computational package is designed to be stored up in high memory while Applesoft BASIC programs are running, but it does take up about 9K of RAM. The result of the computations is a stream of data which still must be fed through a Digital to Analog Converter and then amplified to drive a speaker. The package from Don't Ask Computer Software or from Tronic includes the SAM software, the DAC and the amplifier.

Voice Recognition

The technical challenges involved in getting an Apple to understand what you are saying to it are substantially different from those involved in speech synthesis. In both cases, you are concerned with storing the digital representation of words in as compact a space as possible; however, the difference is that you need to store the word in a way which makes it easy for the Apple to recognize while for synthesis you need to store it in a way that will later make it easy for a human to recognize.

To achieve good speed and minimize computational requirements, voice recognition systems ignore the phoneme elements and ignore the bases and origins of the various harmonic components in a voice. The objective is to capture an entire word and to treat its total waveform sequence as a single, hopefully recognizable, shape.

In all of the Apple systems the human who will be using the device speaks in a short list of words through a microphone. These words are stored in RAM as if each one was a shape. Later, when you want the system to accept a verbal command, it records your command and then runs back through the list it stored earlier to see if your command looks like any of the stored words.

Data Compression

Any system which is based on the direct digital recording of speech can get into memory trouble quickly because an accurate recording requires a data flow of about 8000 bytes per second. To store 40 words at this rate requires 240K of RAM, which is out of the question. The LPC method of speech synthesis is based on recording at this data rate and then using

an elaborate computational model to compress this data down into about 100 bytes per second. Because these compressed words must be reconstructed later, the computational coding task is daunting.

Speech recognition systems also require compression, but since there is no need to reproduce the speech later, the approach to compression can be far less elaborate. The standard tool in compression for speech recognition is a "frequency spectrum analyzer." This is a device which effectively is made up of an array of waveform detectors, each set to a different frequency within the range found in voice. Each one measures the amplitude of the waveform components in its own range. When you run a waveform through one of these devices, it produces a report on the frequency spectrum of the input.

The Voice Input Module (VIM) from Voice Machine Communications (see Figure 11.2) collects in these frequency spectra at the rate of 200 per second while a word is being spoken. It uses a 16 channel analyzer with eight bit precision in each channel, so 16 bytes of data are collected in each sample.

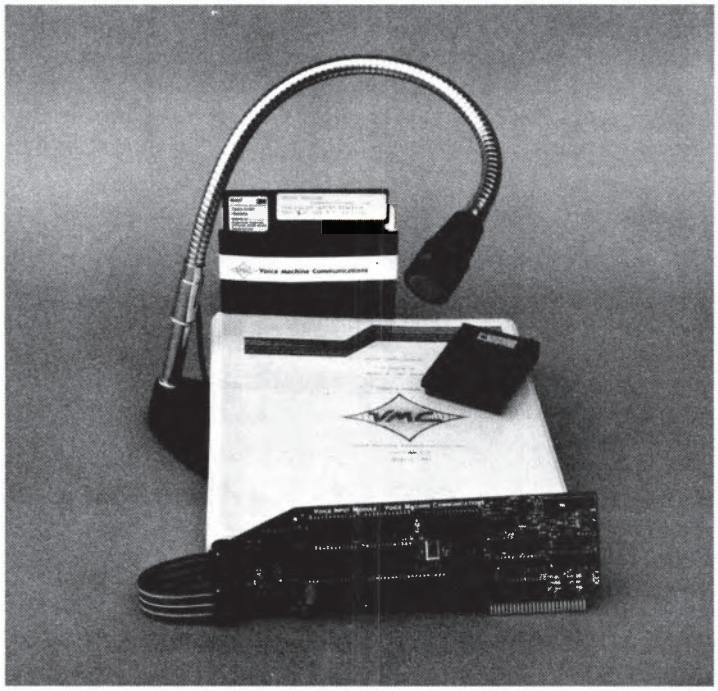


Fig. 11.2 Voice Input Module from Voice Machine Communications. The system is available with a gooseneck or a remote microphone.

This input rate of 3200 bytes per second is still very fast, so there is a further compression step. Each spectrum is treated as a bar chart with 16 bars. This chart is submitted to "spectral slope encoding" in which you simply go from bar to bar and if one bar is higher than or equal to the previous bar, you record a one, but if it is lower, you record a zero. This results in a 16 bit number to replace your 16 byte spectrum, so you've cut your storage requirement to 400 bytes per second.

Obviously, this ignores a great deal of the information in the spectrum, so you make a second compression pass with a different technique to get an abbreviation of a different aspect of the

spectrum. In this process, called "spectral energy encoding," you draw a line through the average height of the bars and assign ones and zeros according to who's above and who's below the line.

In fact, the process continues along these lines, further cutting and mashing, until you've gotten things down into the range of about 100 bytes per second of speech. This is similar to the degree of compression achieved with LPC, but it is done much more quickly in a rapid series of abrupt strokes. In the case of the VIM the end result is that you can store 80 words in 8K of RAM and thus, remarkably enough, the device is able to use these mangled and compressed relicts to accurately and rapidly recognize what you are saying.

Performance

In addition to the VIM, there are voice recognition systems available from Artra (Waldo), Scott Instruments (Voice Entry Terminal), and Voicetek (Cognivox). The Voicetek system is based on a much simpler compression method called "delta modulation." It doesn't do extremely well on recognition, but it is also a speech record and playback system so it is unique in using the same inexpensive chip as the basis of a system to let you teach it what to say as well as teach it what you mean.

Waldo is not extremely ambitious with regard to quality of recognition, but you can teach it to recognize 24 phrases and it is the only one of the systems which will recognize your commands from a distance of six feet from the microphone. However, the really remarkable thing about Waldo is the umpteen other features squeezed onto the card. Waldo has a real time clock, a noise generator, and a BSR Controller which lets you control light switches and appliances throughout your house just by talking to your Apple. In addition, you can add a Votrax SC01 chip for rough quality speech synthesis or a TMS 5220 for high quality speech with a fixed 206 word ROM-based vocabulary. You could thus give Waldo a verbal instruction to say something to you and then whistle at precisely 8:35 p.m.

The Voice Entry Terminal (VET) from Scott Instruments and the Voice Input Module (VIM) from Voice Machine Communications both emphasize high accuracy in voice recognition. The VET has an external cabinet with 16K of RAM and can hold 40 words for recognition while the VIM fits entirely onto a single card and squeezes 80 words into 8K of its own RAM. In both of these systems the idea is to have the device recognize your instruction and respond by sending ASCII character codes into the Apple keyboard input port. As a result, the Apple itself is completely uninvolved in the acquisition and recognition process, and it accepts the characters just exactly as if they were typed at the keyboard.

Using a Voice Recognition System

To use the VIM or VET, you first go through a training routine where you teach the device the words it will have to recognize. In this process you speak the word into a microphone five to seven times. With each pass, the compression system processes the input and compares it to the previous attempt. Thus, it gradually accumulates the best approximation of the average way in which you pronounce that word. You repeat this process for each of the 40 (VET) or 80 (VIM) words you want it to recognize, and when you are finished with the training session, you store a copy of the set of compressed words on disk for future use.

You can create as many sets as you like, but only one set can be loaded and active at a time. Each person who will use the system must have their own personal word sets, because these devices can only recognize commands from the one person who originally trained them. Further, you may have to retrain the system from time to time as your own voice changes slightly from month to month.

Once installed, trained and running, the VET and VIM are still sensitive to changes in your exact pronunciation of a word. In effect, you are being trained as much as the device is, since you must learn to say a word in nearly the same way every time you use it. Particularly for the VET system, you will do best if you carefully choose words or phrases which are clearly distinctive from one another. The VET or VIM can be instructed to match any string you like with a given word and so you can use longer distinctive terms in the place of the similar sounding shorter words you need to have typed into the Apple.

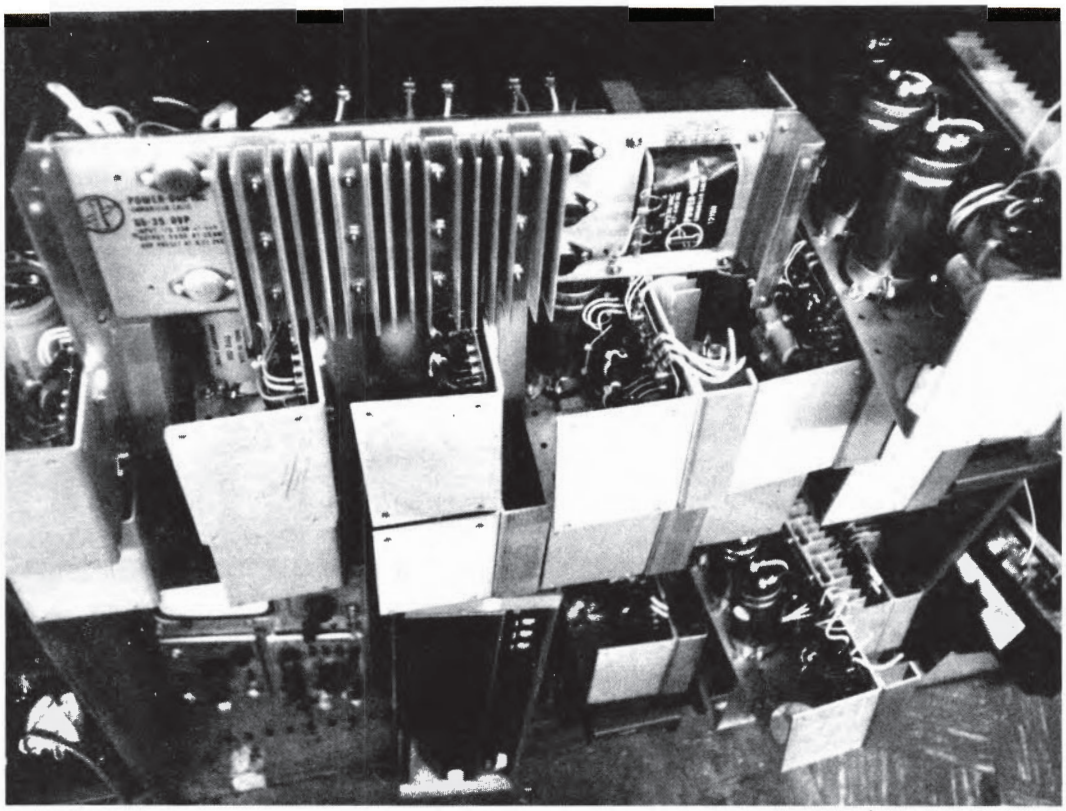
For a well motivated speaker such as a handicapped person, it is possible to achieve nearly 100 percent recognition of your commands with the VIM system. Most users who don't use the VET or VIM on a daily basis will find that some words are difficult to communicate to the device without careful attention and some repetition. Thus, these systems are a wonderful boon to the handicapped and to people who need to have their hands free while issuing commands (in a low noise environment). In addition, if you use them on a regular basis you will become sufficiently well trained that the recognition system will be very reliable and easy to use.

It is clear that a great deal of the potential power of these devices has been sacrificed in order to save on expensive RAM memory. However, the dramatic plunge in the cost of RAM over the past four years as well as the increased availability of inexpensive, high speed microprocessors will make it possible for the people who designed these pioneering systems to develop much more versatile products in the near future.

PART 3

The Circuit Apple

- **CHAPTER 12 Electricity and the Power Supply**
- **CHAPTER 13 Electronics and Apple ICs**
- **CHAPTER 14 Switches, Counters and Converters**
- **CHAPTER 15 Laboratory Data and Control Systems**



Chapter 12

Electricity and the Power Supply

The Electric Apple: The Power Supply

Electricity is used both as a means of transmitting energy or power and as a means of transmitting information. The DIP chips which make up the Apple's circuitry specialize in the information side of things and are usually referred to as "electronic" devices. However, work requires energy, and in order to operate, these electronic devices need a steady supply of energy. It is the business of the power supply to provide that energy in exactly the form the chips will need.

The Apple's power supply is the modern day descendant of a long tradition of electrical devices reaching back for over two centuries. It is a very sophisticated electrical device however and it performs a variety of feats which would have been unthinkable 20 years ago.

The principal duty of the power supply is to collect energy from the wall outlet as it is supplied by the power company and to convert it into a form that can be safely used by the electronic components. Its second duty is to maintain a constant surveillance of the electronic circuitry and to attempt to minimize the damage when it senses a catastrophe coming on.

Measuring Power

There are several different terms used to describe amounts of electricity. The one you hear about the most often is "voltage." Voltage does not describe the amount of electrons flowing. Rather, it describes a force field that can cause action at a distance—a rather science fiction-like concept.

Voltage is to electrons as a magnetic field is to iron filings; it is able to make them move. Place an electron in a high voltage, attractive electric force field and it will zoom towards the point of greatest attraction, splatting against the field source with a force proportional to the voltage which attracted it. Of course, if you set things up like this, all you'll end up with is a source of a force field with a lot of electrons splatted up against the side of it. Sometimes this sort of thing is useful, but it is also a reasonable description of what happens to you while you're scuffing your feet on a carpet and accumulating a static charge.

Charge is a way of describing how many excess electrons have piled up somewhere. We'll be coming back to this later because it is the central concept in the design of most RAM memory chips, but for now let's assume we don't want to accumulate any electrons. The conventional way out of the problem is to set things up so that electrons are attracted to one side of a force field and repelled from the other side. For every one that comes in, another one gets spat out the other side. All that you have to do is to connect a copper wire to one side of the force field generator and connect the other end of the wire to the other side. When the force field is turned on, electrons will seem to be flowing in a continuous circuit but will never accumulate anywhere.

So where do all these electrons come from? Actually, they're just sitting there in the copper wire being part of copper atoms when nothing else is going on. When some of them are participating in the circuit flow, the copper atoms don't really notice since every electron they lose gets replaced by another one from somewhere else in the wire. It is because of copper's rather blase "no sweat" attitude toward trading electrons among neighboring copper atoms that it is constantly being put into wires, strung between telephone poles, and called a "conductor."

Current

How fast does an electron travel in a copper wire? Very slowly, actually. The situation here is much like one in which you lay out a row of billiard balls in a line, all touching each other. You take your pool cue and, simulating an electric force field, you tap the ball closest to you. The ball at the other end of the row rolls away almost instantly, but the several balls along the row barely move at all. It is the impulse which has been transmitted rapidly down the line. Similarly, the force field is transmitted along the wire at a pace somewhere near the speed of light, but the electrons at any given point just rattle around a little bit as the force gets passed along essentially undiminished.

The stronger the force (i.e, the higher the voltage), the more electrons move and the faster they move, but it's still a fairly slow passage. The term "current" describes the number of electrons that actually pass by some point along the wire in a given interval of time. The current is related to the magnitude of the voltage pulling them along, and it is also related to how much bouncing around they do as they pass from copper atom to copper atom.

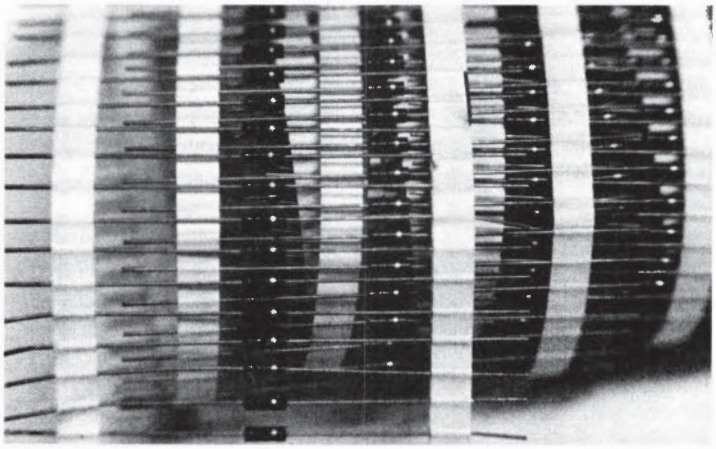


Fig. 12.1 Roll of resistors.

Resistance

If you placed a tube full of powdered carbon between the two sides of the same force field source, the electrons would move much more slowly. The way electrical people describe this property of powdered carbon (and a lot of other things) is to say that the substance has high "resistance," i.e., it resists the flow of electrons. Many useful electric circuits include short stretches of copper wire conductor linked together by short stretches of these powdered carbon tubes called "resistors" (see Figure 12.1). The higher the applied voltage in such a circuit, the higher will be the current flow, but as you add more and more resistors, the amount of current flow will decrease.

The power available for doing work is related to the amount of current flowing. By adjusting the strength of the force field (voltage) and the amount of resistance (measured in ohms), you can produce just the amount of current flow (measured in amps) that you happen to need at any given point in the electronic device you're trying to operate. It is important to limit the current flow and to see that it doesn't get out of hand. You see, as the electrons bounce along, even in a good conductor like copper, they "rub" against the fixed atoms and make the wire get hot. This may be fine if you're building a space heater or a brilliant electric lamp, but when you're dealing with incredibly tiny and fragile microelectronic circuitry you had best keep the current carefully under control.

Alternating and Direct Current

The form in which electrical power is supplied from your local water fall or nuclear reactor is as a constantly varying voltage which reaches a maximum of 163 volts, then drops to -163 volts, comes back up to +163, etc. If this is connected to a circuit, the electrons alternately start moving toward the power plant and then away from it. This is called Alternating Current (AC). The voltage varies smoothly from its minimum up to its maximum and back 60 times a second.

This sort of electric power is useful for making light bulbs glow because it's really the friction that's important. This is much like rubbing something to make it heat up—the fact that you rub back and forth doesn't decrease your effectiveness in the slightest. When the current flowing in the wire is evaluated it is approximately equivalent to what would result from a steady voltage of 115 volts. The formal name for this "effective" voltage is the Root Mean Square (RMS) voltage; this reflects how the 115 is derived mathematically from the 163.

If you glance at the Apple power supply near where you plug in the cord, you will see that it says "115 VAC, 60 Hz." This is Apple's way of specifying what it expects the local power company to be providing; alternating current with an effective force of 115 volts varying at a rate of 60 full cycles per second. A different power supply is used for European Apples since European power companies have chosen to provide 220 VAC at 50 Hz.

The Task of the Power Supply

OK, so if there's all this power available, why does the Apple require a large box called a "power supply?" The problem is that if you connected the power company's electricity directly to the motherboard, you'd probably just start a fire. 115 VAC is the source of the Apple's power, but this is highly toxic stuff for electronic components. Unlike light bulbs and space heaters,

high current and strong voltages are not the name of the game. Most electronic components being built these days require a steady, unvarying five volt power supply. Some older chips may require any or all of +5 volts, -5 volts, +12 volts, and -12 volts. Few of the chips on the Apple II motherboard can make do with only +5 volts, and even the //e and //c require little bits of -5, +12, and −12.

What the power supply does is to transform current at 115 volts into varying current at each of +5, −5, +12, and −12 volts, and to coax the variations out so that the various voltages each emerge as smooth, steady constant Direct Current (DC). This smoothing process is accomplished in part by devices called "capacitors" which store charge while the voltage is increasing and then dump it back into the circuit while the voltage is decreasing. It takes a few seconds for these capacitors to drain fully in the //e and this is why the memory does not clear completely the instant you turn off the power.

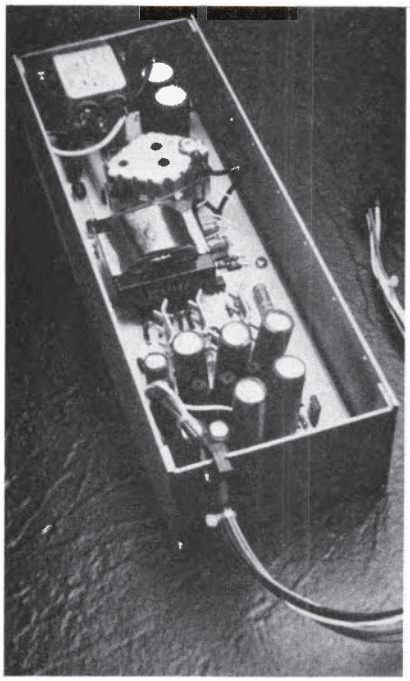


Fig. 12.2a (above) Inside the Apple power supply for the II/II+ and //e.

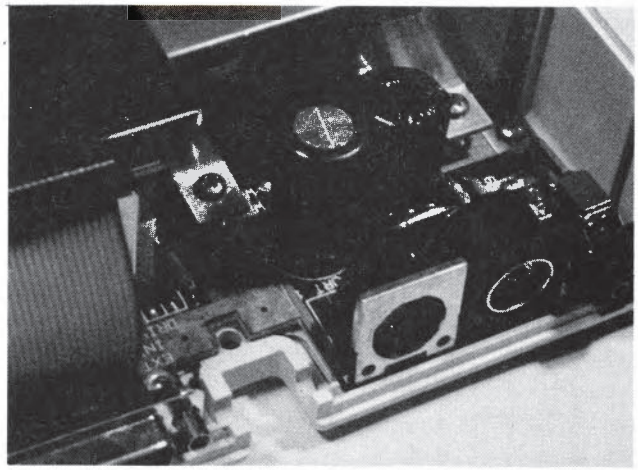


Fig. 12.2b Just behind the //c power connector, a large capacitor and inductor begin filtering the incoming power.

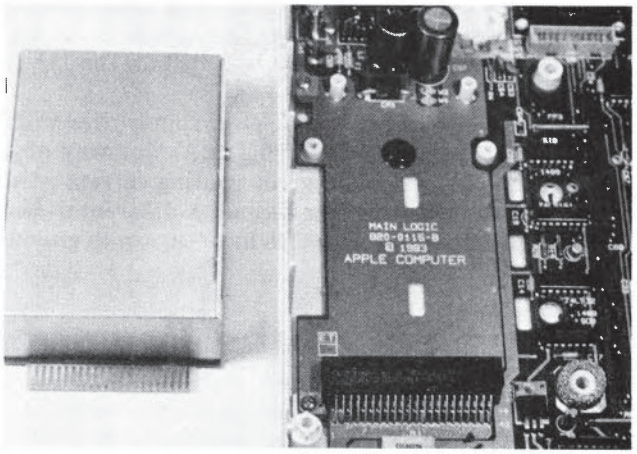


Fig. 12.2c (right) The Internal Power Converter plugs into the //c's only slot. It filters, smooths and regulates the incoming power and sends protected +12, −12, and +5 volts to the motherboard.

Transformers and Switchers

The process of transforming the voltage is performed by an electrical component called a "transformer." In many power supplies the transformer is a huge, heavy object that becomes very hot as it works. Wozniak was determined to make a small, light, cool, power supply that could be safely used inside a plastic cabinet without a fan, so a traditional transformer just would not do.

The Apple's power supply does have a small transformer, but some modern electronic trickery is used to make the transforming process extremely cool and efficient (see Figure 12.2). The efficiency of a transformer is directly related to the speed at which its alternating current is alternating. In the Apple power supply, the 60 Hz AC power is first smoothed to steady DC, and then a fancy system of components is used to make it alternate once again but at very

Fig. 12.2d (right) Inside the //c Internal Power Converter box are a number of capacitors and transformers and the high current transistors used in the switching regulation system.

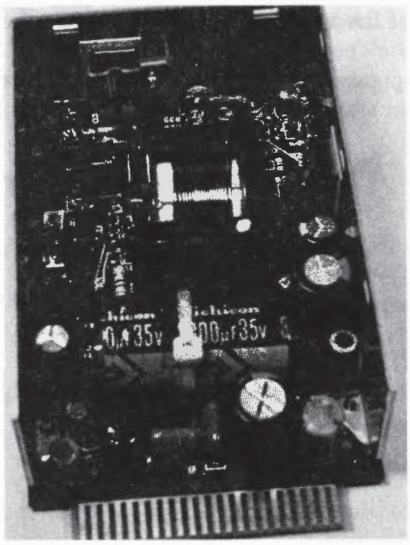
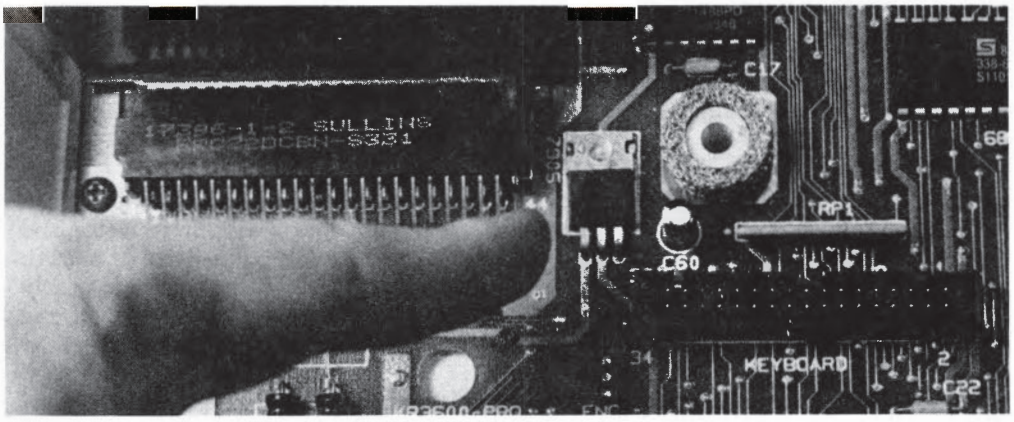


Fig. 12.2e (below) The 7905 voltage regulator chip on the //c motherboard converts –12 volts into a fourth, –5 volt supply.



high speed. This high frequency power is then fed through the transformer system. This type of power supply is called a "switching power supply." They are fairly common these days (though not at all universal), but Wozniak's choice of a switching supply was bold and daring at the time.

In the //c, the first step of smoothing the 60 Hz AC power to steady DC is done outside the Apple in the External Power Supply. This external supply sends a steady DC current at +15 volts to the Apple. The Internal Power Converter (see Figure 12.2c and 12.2d) accepts this 15 volt DC input and then uses a high speed switching system to produce the +5 VDC, +12 VDC, and -12 VDC which it sends to the motherboard. The -5 VDC needed by some components is derived on the motherboard by a 7905 voltage regulator (see Figure 12.2e) working on the -12 volts supplied by the Internal Power converter.

The Internal Power Converter will actually work perfectly with inputs ranging from +9 volts DC to +20 VDC. Thus the //c has been designed to operate from the power available from 12 volt automobile batteries and other such sources. This is particularly important in various parts of the world where steady 120 volt AC power just can't be relied on. In addition, it adds to the //c's portability since it can be conveniently connected to a rechargeable battery or even to the cigarette lighter in your car. If, however, the power drops below nine volts, the "power" light on the keyboard will begin to flash and you must turn off the Apple immediately.

The Apple's Crowbar: Limits on Available Power

The voltage source in the the power supply may be thought of as a pool of potential energy which is constantly being drained off as the current it causes is used to do work. The power supply is carefully designed to use the 115 VAC it has available to steadily replenish the potential energy pool.

The +12 volt supply is used to power the motor on the disk drive, so when the drive turns on, there is an abrupt increase in demand for energy. The potential energy pool starts to drain too fast, and all the voltages begin to drop. The power supply is designed to sense the increased demand and to respond almost instantly by increasing the frequency of the AC power being fed into the transformer. This yields an increased efficiency of energy supply, the pool is refilled more rapidly, and the voltages are maintained. This process is called "regulation," and it is extremely important for the correct operation of the electronic components.

However, the power supply does not have an unlimited ability to increase its output of current. Apple made an estimate of how much power they thought would be required by a fully equipped Apple II and built the power supply accordingly. These maximums are not quite what they could be.

Each additional peripheral card plugged into the Apple's slots increases the demand for power. With four cards installed, the power supply can just barely handle one disk drive motor, and it becomes possible to cause intermittent failures of the power supply if a drive is kept in continuous use for over 20 minutes. If you put in eight large cards, it may become nearly impossible to get accurate information from the disk drive. Each time the drive turns on, the voltages start to fluctuate and the microprocessor and the memory can get very confused.

In fact, most of the components in the supply can handle higher currents, and the actual limit was chosen to minimize heat generation and thus avoid the need for a fan. Some electronics whizzes have picked out the few circuit elements in the Apple supply which are the most

sensitive to heat, changed them, and added heat sinks, with the result that their supply can handle up to twice the planned rating.

Normally, though, if the power supply experiences sudden and excessive increases in demand, it is designed to shut down completely. This design feature is intended as an important protective feature and it has saved thousands of Apple owners from thousands of dollars worth of damage to their systems. However, to make the system as safe as possible, Apple has adjusted this shut down feature to occur very suddenly as soon as voltage slips just a bit out of expected range.

The device that causes the shutdown is called a "crowbar," in part because it responds to a sharp but modest change in voltage by producing a massive change in the operation of the power supply and in part because its electrical symbol on schematic drawings resembles a crowbar. Its formal name is Silicon Controlled Rectifier (SCR).

Two Apple Frying Events: Producing a Short Circuit

You will recall that current is related to both voltage and to resistance. This can be represented mathematically by a simple formula, I = V/R (where I is the standard symbol for current). This formula predicts that if for instance your circuit has a 12 volt supply and 24 ohms of resistance, then the current will be 12/24 = 0.5 amps. The only reason to start getting mathematical about this stuff is because it is worthwhile to point out what happens if the resistance drops to somewhere near zero. If you have a nearly infinite pool of energy (such as a nuclear power plant) supplying the voltage, the formula predicts that a zero resistance circuit will have a current flow of nearly infinity. At amperage in this range, copper either melts or just simply evaporates.

This sort of event is called a "short circuit." It occurs whenever a voltage supply line is connected to a returning "ground" wire without any resistance separating them. The Apple's power supply is not able to provide spectacular amounts of power, so a short in the Apple won't evaporate copper, but it can do an awful lot of damage. Apple users have two favorite ways of producing a short. One of these occurs when people disobey all warnings and attempt to plug in a new peripheral card while the power is on. If the card comes in at the wrong angle, pin 25 connects to pin 26 and disaster strikes.

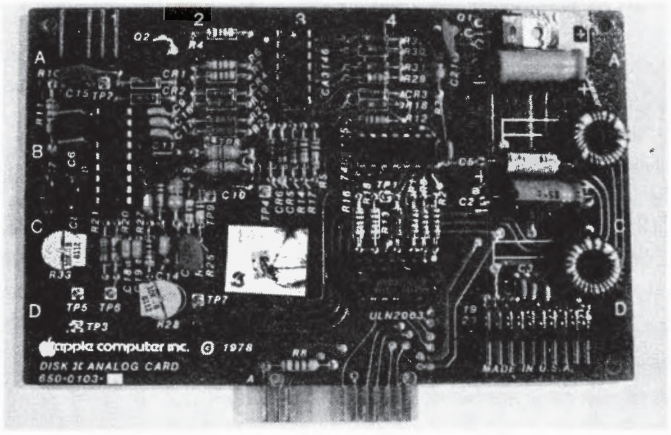


Fig. 12.3 Analog board form inside the Disk II drive. The chip marked ULN 2003 can burn out if a drive connector is plugged into the interface card incorrectly. This happens frequently enough that Apple service personnel call this chip the circuit breaker.

The other is a delayed action effect that comes about when the disk drive connector is plugged into the disk drive card in the wrong direction. When you turn on the power, a chip inside the drive called a ULN 2003 (see Figure 12.3) burns and smoke comes out of the drive. This is an interesting failure because, for a variety of reasons, the power supply can't detect the fault and the only symptoms are the smoke and the fact that your disk drive won't work anymore. Fortunately, this isn't much of a disaster since the chip only costs 40 cents, nothing else gets damaged, and your dealer can replace it in just a few minutes.

Otherwise, the crowbar system is very effective at preventing excess current flow by keeping up its surveillance of very sharp voltage changes. The Apple power supply system is well designed and reliable, but there are several reasons why you may want to purchase enhancements or replacements.

Replacing the Power Supply

Power Supply Failure

Apple //e owners are accustomed to using the Control/Open Apple/Reset sequence to clear the system and restart things, but Apple II and II+ owners are constantly reaching back and toggling the on/off switch. This mechanical switch has the highest failure rate of any component in the Apple and when it goes, you ususally have to replace the entire power supply. Just behind the switch in failure rate is the power supply itself. The supply in a heavily taxed two- or three-year-old Apple may wake up one morning and quietly decide to expire.

If you don't have too many cards in your Apple, you should probably have the supply replaced with another standard Apple power supply. Your dealer will do this for about $100. If, however, you have a lot of large cards in your Apple or are contemplating getting some, you should give strong consideration to purchasing a power supply from some other company which has a higher maximum current output.

Some expansion modules for the //c draw their power from the //c's internal power converter, and they can easily overwhelm it. There is no convenient way to change the //c's power supply, so all power-hungry expansion modules must come with their own power source.

Enough Power

An overtaxed power supply usually reveals itself as cryptic and bizarre behavior of the computer including nonsense reads from disk, or inexplicable "hanging" of the system during disk accesses. There are a variety of things which can cause these symptoms so you'll have to do a little checking before arriving at a verdict. The simplest approach is to remove a few cards (such as a printer or communications card) and see if the problem clears up. If it does, then either the card you removed is bad or it's your power supply. You can usually choose between these two possibilities by testing the suspect cards standing alone in your computer. If they work fine, it may very well be a power supply problem. Unfortunately, most Apple peripheral manufacturers don't tell you how much power their cards need so you're left to using this sort of test.

The Apple II, II+ and //e supplies are rated in terms of their total output, but for the //c, Apple specifies the amount of power available on its external connectors. This is the total power minus what the //c itself uses.

Total output ratings for the II, II+ and //e:

+ 5 volts	2.5	amps
+12 volts	1.5	amps continuous
	2.5	amps for intermittent use for 20 minute periods
− 5 volts	0.25	amps
−12 volts	0.25	amps

Expansion power ratings for the //c:

+ 5 volts . . . . . :	1.5	amps
+12 volts	0.6	amps continuous
	0.9	amps intermittent
− 5 volts	0.05	amps
−12 volts	0.10	amps

The biggest problem is with the +5 volt supply. The Apple motherboard uses up nearly 2.0 of the total 2.5 amps. Apple warns you that a grand total of only 0.5 amps is available for all other cards. A card that is plugged in will draw power whether or not you are using it, so there is no simple way around this problem.

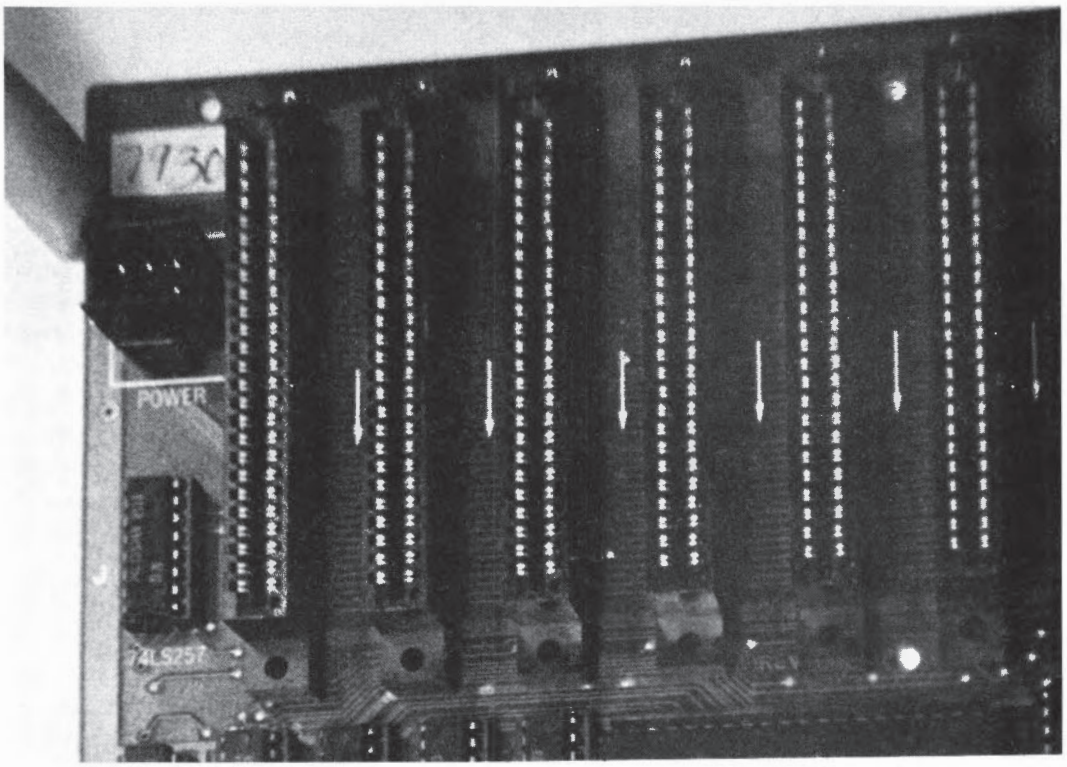


Fig. 12.4 The power connector on the II+.

Replacement 5 Amp Internal Supplies

Several companies sell switching power supplies with crowbar current limiting protection which provide increased amperage at the various voltages. These supplies fall into two categories. One set looks identical to the Apple power supply. To install one of these, you turn off the Apple, unplug your supply from the motherboard (see Figure 12.4), remove four screws on the bottom of the Apple and then reverse the process to install the new one. These supplies typically provide a full 5 amps of +5 volt power. Once the Apple has used up its 2 amps, you still have 3 amps left over for your peripherals, a six-fold increase.

One concern when you provide increased current is that your power supply will generate increased heat. This doesn't seem to be a serious problem for supplies that are properly designed. Further, if you have so many cards that you need an increased power supply, you really ought to have a fan (see below).

The real problem with these internal replacement supplies is that no well established Apple peripheral manufacturer makes them. Many individual supplies have erratic performance when current flow is at its highest. This means that they fail when you need them most. The symptoms include odd behavior linked in time with the extra demand due to startup of the motor in a disk drive. The West Coast electronics supply houses which sell the internal supplies (JDR, Concord Computer Products) are all essentially retailers rather than manufacturers. These supply houses have some stake in maintaining your confidence, but this stills leaves you buying Brand X to plug into your Apple.

External Replacement Supplies

The other approach to enhancing the power supply is to use an external power supply which has a cable that reaches in to the motherboard power connector. This is very easy to install. Just turn off the Apple, unplug the Apple supply, and plug in the new one. Some Apple users dislike the idea of having an extra external box, but it does completely remove this heat source so the inside of your Apple stays much cooler.

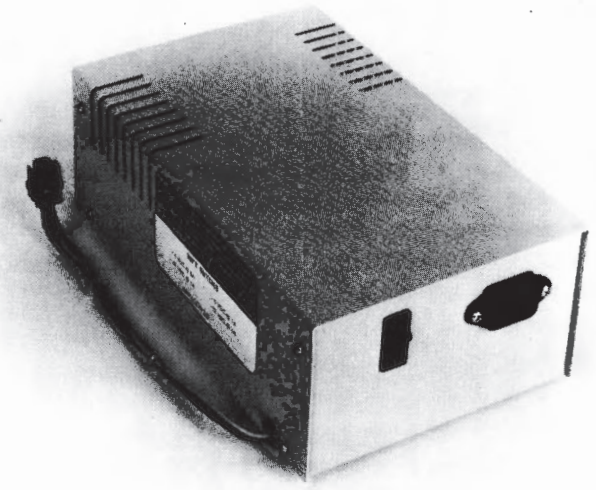


Fig. 12.5 Sup'R Switcher replacement power supply from M&R.

The other advantage is that one such device is available from M&R products as the Sup'R Switcher (see Figure 12.5). M&R is a well established Apple peripheral manufacturer with a variety of products. The supply provides a full six amps of +5 volt power. You can leave the old supply in place but disconnected from the motherboard. This makes installation a little tricky on the //e simply because none of the openings on the back panel are large enough to let the connector pass through. This is easily solved by first removing the Apple supply, passing the cable in through the large opening that action provides, and then replacing the Apple supply.

Some manufacturers of large power hungry boards help you out by providing external power supplies which plug directly into their card. You get such a supply with the Saybrook 68000 processor card and the Metamorphic systems 8088 (see Chapter 30).

The most expensive solution is to purchase an expansion chassis from Mountain Computer. This is a device which provides eight additional slots for the Apple along with their own supply. The Mountain Chassis will be discussed in detail in Chapter 22, but it is enough to say here that most of the large, sophisticated and power hungry cards that cause the problems either won't operate properly that far from the 6502 or are physically unable to fit into the Chassis. This device will solve the power problem for a variety of applications, but its list price of $750 suggests you should pay careful attention to the Chassis' idiosyncrasies as described in Chapter 22.

Sufficient Supply from the Power Company

The Apple II, II+ and //e Power Supply and //c external supply expect to receive clean steady constant power at 107 to 132 VAC. The actual amount of power is measured in Watts (W). The amount of power consumed is related to the voltage your device is using and how much current you are using at that voltage. Simply put, W = VI; the power is equal to the voltage (V) times the current (I).

For the Apple, (5 volts) times (2.5 amps) equals (12.5 watts), and you also have to consider (12 volts) × (2.5 amps) equals (30 watts). The other supplies are fairly minor, so 45 watts is a good rough estimate. The power supply itself requires a fair amount of power to do its work, so the incoming power line is required to provide about 110 watts. This is reflected by the label on the back of the supply that says "1A" (for one amp). To run an Apple you need (110 volts) times (1 amp) equals (110 watts), or the approximate equivalent of a single light bulb.

The voltage rating of 107 to 132 VAC is Apple's way of saying the computer will not be affected by the more modest dips and rises in the power lines. As power use increases and decreases during the day, power companies sometimes have to temporarily adjust their main output voltage by a few percent. A 10 percent drop from 120 VAC on down to 108 VAC is sometimes called a "brownout," but will probably not cause any trouble.

Transients in the Power Line

Most power companies in the United States do a very good job of providing 107 to 132 VAC at 60Hz; blackouts are rare, and the low power requirements of the Apple certainly present no problems; however, a lot of things can happen to your power on the way to your wall outlet and some of these things can be troublesome. The great power supply networks in the United

States were designed by people who expected to be supplying light bulbs, toasters, refrigerators and electric motors. When you are dealing with such devices, there is plenty of room for fuzziness around the edges. The average power supply network is not designed for the needs of complex electronic equipment and there is no major plan on the part of any power company to try to change what they supply along these lines.

In most places around the country, monitoring of incoming power reveals occasional "transients" which fall into the categories of sags, surges and spikes. Sags are those momentary plunges in voltage which make your lights dim for an instant. These are often caused by the huge rush of current into large electric motors somewhere nearby. Particularly in the //e, the large capacitors and the voltage regulation circuitry built into the Apple power supply will pull you through most of the time without any effect. However, if this is an ongoing and serious problem, you may have to invest in an "uninterruptible power supply" (see below), since no amount of inexpensive filtering and adjusting will be much help.

Surges and Spikes

Surges and spikes are both sudden, sharp increases in voltage on the line, but the term "surge" implies long duration (a few tenths of a second to a second) and modest voltage (200 to 300 volts), while spikes may come and go in a few millionths of a second but achieve maximums of many thousands of volts. Surge protection requires a device which can dissipate the large amount of excess power that is pouring in. Spike protection requires a device that can respond at extremely high speed before any of the voltage increase can pass into the computer. The specifications on protection devices often refer to a "two stage surge suppressor" and this actually means that the manufacturer has thrown in a few components for a high power surge section and few components for a high speed spike section.

Regulating Through a Surge

The surges are an artifact of the kind of equipment the power company uses to maintain a steady output level in its network. Their systems are perfectly capable of cutting output when demand drops, but most systems use "elctromechanical relays" (see Chapter 14) which take anywhere from a few thousandths to a few tenths of a second to carry out switching commands. While everyone is waiting for the switch to throw, a surge is rising throughout the local metropolitan area.

The Apple's power supply actually handles these fairly well all by itself because of its well designed internal voltage regulation features. In fact, the real problem with a surge is that it can overwhelm and destroy the fancy spike protector you purchased for your Apple. This is why equipment from RH Electronics and EPD includes indicator lights and fuses to warn you if your spike protector has been fried.

Switches, Sparks, and Spikes

Most troublesome among all of the problems are the spikes. Unfortunately, these have a variety of causes ranging from the events which occur when you throw a wall switch to distant lightning strikes. When current is flowing in a wire, it sets up a magnetic field in the space around it. When you throw a switch to stop current flow, the fields collapse and power rushes back into the line. In the very first few instants after the switch opens, the two switch contacts are extremely close together, so as the voltage from the collapsing fields runs up into the hundreds and thousands of volts, you get a high voltage spark across the switch.

Some devices, such as relays, motors and transformers, have large "inductors" which are designed to build up very large magnetic fields. When these are turned off, they can cause a sort of ringing series of sparks across the opening switch. Each spark discharges the rising voltage, then it starts to rise again causing a new spark, all in a repeating cycle until discharge is complete a couple of thousandths of seconds later.

It is possible to design switches in such a way as to prevent this, but your home, office and lab is filled with devices in which this sort of thing happens all the time. One of the worst groups of offenders in this regard are the manufacturers of letter quality printers; people who really ought to know better. In addition, there are lightning strikes and plain old static to worry about. Three thousand volts for a millionth of a second will not even change the glow on your average light bulb, but it has every chance of confusing all hell out of a 6502 or dynamic RAM chip. Further, it can damage or destroy individual little circuit elements scattered about within the computer.

Electromagnetic and Radio Frequency Interference

A second kind of power line "noise" called Radio Frequency Interference (RFI) noise comes from radios, TVs and other computers. This is high frequency, low power noise but can have very annoying effects on an Apple. There is no problem with the voltages or currents involved, but when you have slight variations at rates of 10 kHz (thousands of cycles per second) to 30 MHz (millions of cycles per second), the digital circuits can be fooled into thinking that the variations are actually information they are supposed to be paying attention to. The fact that your AC power lines can carry complex electronic signal patterns is used to advantage by some of the "power line carrier control devices" (see Chapter 14), but otherwise, this capability is a substantial potential nuisance.

One very troublesome aspect of electro-magnetic oscillations at these frequencies is that although they can travel in wires, they can also be broadcast from one wire to another. For this reason, it is important that noise at these "EMI-RFI" frequencies be filtered out as close as possible to the computer chassis. As a result of all this, it has been standard practice for several years for nearly all electronic devices to be built with "filters" to clean up the incoming AC power and also to limit transmissions which might effect other devices.

Since the Apple II's switching power supply generates quite a bit of this kind of noise, Apple had to build a very effective EMI-RFI filter into the machine in order to ensure that it met Federal Communications Commission (FCC) regulations. This filter, fortunately, works very well in both directions so only the owners of very early Apple IIs (which have a less rigorous filter) need be concerned with buying an EMI-RFI filter.

Adding Power Protection

Most Apple owners will never have a problem with the quality of their incoming power, but no one can be completely confident. The real question is whether buying power protection is like buying a fourth lock for your apartment door (a little paranoid) or like buying automobile insurance (you'd be crazy not to have it). Correspondingly, available devices range from $14 to $900.

There is no convincing proof that everyone needs one of these extra protection devices, but the idea of a lightning bolt zapping your Apple is pretty arresting. A very large proportion of

Apple owners have, in fact, purchased some such device. These things may not do too much for your Apple, but they do give you a warm positive feeling that you've taken firm and constructive action, in advance, to prevent a whole host of imaginable future mishaps.

Power Protection Devices

Amperage of Multiple Outlet Boxes

One advantage provided by many of these devices is that they usually include several outlets. Not only is this a convenience for plugging in your computer, your monitor, your modem, etc., but it is important to plug in all the parts of your computer very close together to get a clean common ground point. The easiest way to do this is to use a box with several outlets.

The Apple needs one amp, and your monitor, printer and external modem may together require two more amps, so an outlet box rated at four amps, such as the System Saver by Kensington Microware, is calling it a bit close. If you plug in a hard disk, the power coming out of the System Saver becomes a slashed up mess since the device thinks it's in an overvoltage situation.

Surge Protection

For most of the less expensive protection devices, your first priority is to see that the device will respond in a few billionths of a second to shunt voltage spikes away from the Apple. All of these devices work by effectively watching the magnitude of the incoming voltage and throwing a switch when the voltage goes above a certain level.

As long as the thing works fast, the actual "clamping" voltage is not crucial: 200 volts for five billionths of a second is no big deal in terms of power effects. The protection renders the spike into some low power RFI noise which can be filtered out by the Apple.

The principal way to build a "switch" that can tolerate thousands of volts but throw open in a few nanoseconds is to use a device called a Metal Oxide Varistor (MOV) which is a special application of materials technology and transistor design that is essentially made to order for the job. Unfortunately, the actual physical layout of the components in the surge protector can slow the response of an MOV. For this reason, the "Zener Ray" option on the RH Electronics Super Fan II (see Figure 12.6) relies heavily on a carefully designed filter system to catch very high speed events, and then follows with a high speed "zener" diode all before the MOV has to come into line.

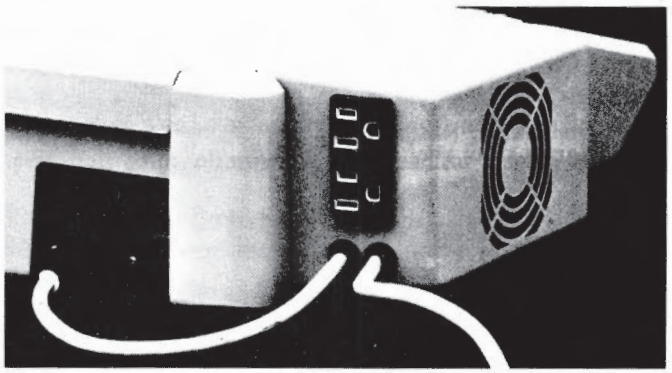


Fig. 12.6 Super Fan II from RH includes two extra outlets, surge protection and a front facing on/off switch for the Apple.

The RH Electronics design also points up a problem in judging these devices from their published specifications. The components used by a manufacturer may have excellent properties, but when you are working with high speed signals, only artful electronic design can tease high performance out of the components. Many manufacturers publish the specifications for the components they used, rather than the specifications for their own actual, assembled products.

Popular surge protectors for the Apple are made by EPD (Electronic Protection Devices), RH Electronics, and Kensington Microware. The EPD surge protector has a power section which can dissipate 10,800 watts for one millisecond (10.8 joules) and a spike section with five nanosecond response. This same surge protector is used in nine of their products. The Kiwi is a one plug device, the Lemon/(EC-I) includes six outlets, and the Lime/(EC-II) comes on the end of a six foot cord instead of plugging directly into the wall socket, and it also has an on/off switch. The Electro-Clamp (EC) versions differ only in that they come in white as opposed to glaring green or yellow.

Both RH Electronics and Kensington Microware offer their surge protectors as part of a device with a fan, a switch and a second power outlet (see Figure 12.6). The comparable ratings on the RH Electronics Super Fan II w/Zener Ray Option are 50 joules of power dissipation for surges and a very high speed voltage clamp which can begin the system's initial response in a small fraction of a nanosecond. The Kensington Microware System Saver fan has protection rated at 40 joules of power dissipation but a somewhat slow response of 50 nanoseconds. Fifty billionths of a second may not sound slow to you, but in that length of time an arriving spike from a lightning strike may already have sent a 5000 volt pulse into your Apple.

EMI-RFI Protection

The Apple has a very well designed EMI-RFI filter built into the circuitry of its power supply. It can benefit from a little prefiltering, but that's only true if the external filter has better performance than the Apple's own filter. For the most part, external filters are more important for their role in spike protection than in noise filtering.

The Peach/(EC-IV) and the Orange/(EC-V) from EPD resemble the Lemon and the Lime respectively but include EMI-RFI protection. The way this is rated is to cite the frequency ranges which will be filtered out, and to indicate how complete the filtering will be. The range of affected frequencies for the EPD filter is 150 kHz to 30 MHz; however, the filter system is not equally efficient at all frequencies. You are thus told that there is an "attenuation" range of 5 dB to 58 dB. This cryptic notation is given in units of "decibels."

A decibel is just a trendy mathematical way of stating a ratio. It is often used to describe the relative power of two sounds in the equation:

$$dB = 10 \log (P \text{ out}/P \text{ in})$$

However, the same equation applies for electrical power. More pertinently, the designers of filters use the relation between voltage and power (Power equals Voltage squared/Resistance) to produce a decibel equation for voltages:

$$dB = 20 \log (V \text{ out}/V \text{ in})$$

This means that a 5 dB attenuation reflects a ratio of 1.78 to 1, and a 58 dB attenuation means a ratio of about 800 to 1. Thus, for some frequencies, this filter only cuts the noise in half, while for others it reduces it by 800 times. The Kensington Microware System Saver is rated at 20 dB to 50 dB over a frequency range of 600 kHz to 30 MHz.

After you've sorted out these unnecessarily obscure numbers, you'll be disappointed to learn that the performance of a filter is determined by what happens to be plugged in around it. Therefore, the stated specs aren't a reliable guide. There is no authority that does testing on these devices for the Apple, so if you have critical needs (laboratory instrumentation, etc.) you should be prepared to have the testing done where your computer is set up. It is for this reason that the RH Electronics people do not claim frequency or attenuation specs for their device.

Uninterruptible Power Supplies

Blackouts and blown fuses are fairly rare occurrences, but if you keep large amounts of valuable data in RAM memory, or if you are using a hard disk, you might want to think about buying one of the expensive devices which can give you some protection. Some hard disks need to power down gradually and can be damaged by an abrupt power loss, so you may stand to lose more than just a little data.

An Uninterruptible Power Supply (UPS) is a device which is expected to notice when the incoming AC power drops below some minimum and to then kick in instantly with replacement power. Most of these systems are designed to operate for no more than 10 or 20 minutes which is supposed to give you time to save data and do a controlled power down. An alarm is usually included to help make sure you know there's trouble.

Fig. 12.7a With the Guardian Angel uninterruptable power supply, you get six minutes to save files to disk and shut down in an orderly fashion.

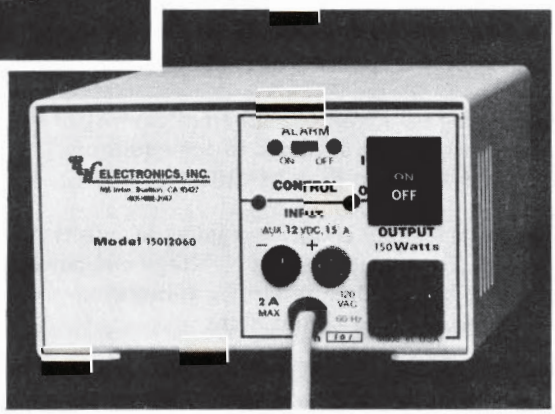


Fig. 12.7b The built-in system in the Guardian Angel can also give you sufficient time to plug in an external 12 volt battery to continue your work.

There is a 150 watt UPS from RH Electronics called the Guardian Angel (see Figure 12.7) which will take over in 10 milliseconds, and then maintain power for six minutes at 150 watts or 15 minutes at 75 watts. For a well stuffed Apple, you'll be pushing towards the shorter duration, but you can get an additional back-up battery. The system includes transient surge protection at all times, so you don't need to buy other protection devices.

EPD makes three Grizzly models for different load requirements. The 200 watt model outputs a square wave, but the 500 watt and 1,000 watt models generate a fully formed sine wave just like the power company's normal supply. All of the Grizzly's are rated at 20 minutes at full load. Before selecting one of these, or the Guardian Angel, you'll have to calculate your total system power needs. An Apple with two drives will do just fine with the Guardian Angel, but 150 watts just won't do it for many larger system configurations.

If you're using your Apple II, II+ or //e in a country where the power supply is subject to frequent interruption and extended failure, these devices may not provide enough storage to keep you going. In that case you'll need a system that uses chopped voltages from a car battery to drive a transformer, and then a means of recharging the battery when the power's on. Better yet, you may want to think about getting a //c which will run directly off of the 12 volts DC supplied by the battery.

Heat in the Apple

All electronic devices generate heat as they work. You can test this out by turning off your Apple (after it's been on for 10 or 15 minutes) and touching the surface of one of the large chips. The heat is not normally a problem. These devices are designed to operate properly at temperatures up to 150 degrees Farenheit (70 degrees centigrade). However, if they are surrounded by stagnant air their temperature will climb above the safe operating range. The first sign of this can be erratic behavior of memory chips—which you will experience as inexplicably bizarre behavior of your Apple, but which starts only after your machine has been on for a while.

The Apple has been designed so that there is a constant "convective" air flow through the box. This means that there is always a gentle breeze through an operating Apple; cool air comes in the bottom, heats up, and expands out the slots on the sides. With just two or three cards in the Apple, convection is more than adequate so long as you don't defeat the design by obstructing the openings in the case.

As you add more cards, however, you are adding more heat generating units and you are obstructing airflow. The convective system gets overwhelmed, and you start getting overheated. Some Apple owners choose to increase convection by operating their Apples with the top removed. This does help a lot and it is very trendy in a "high tech decor" living room, but all it takes is one paper clip bounced in through the open top and you'll get an instant test of the power supply's crowbar shutdown capabilities.

Similarly, the //c is designed to be cooled by convective airflow, but the system only works if you follow certain rules. You must use the flip down handle as a stand (see Figure 12.10) so that the //c does not sit flat on any surface. In addition, you must not place anything on top of the //c while it is running.

Fans

In choosing a fan for a II, II+ or //e, you should consider four factors: sufficient airflow, placement of the fan for best effect, audible noise from the fan and electrical noise from the fan. The M&R Sup 'R Fan (see Figure 12.8) is popular with some Apple owners because it fits inside the Apple, but although it is quiet and unobtrusive, it doesn't generate enough airflow for very overstuffed Apples and, much worse, it generates electrical noise which can "hash" your video screen.

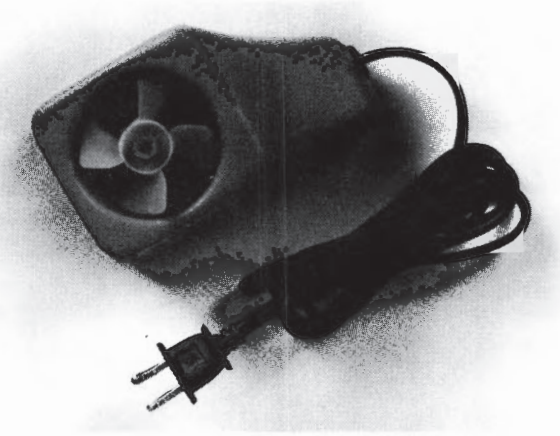


Fig. 12.8 Sup'R Fan from M&R.

For the great majority of Apples needing fans, either the Super Fan II from RH Electronics or the System Saver from Kensington Microware would make a good choice. These fans sit outside the Apple and draw air out over the power supply. This is important because the power supply is often the hottest object in the Apple, and outgoing air should pass over it last to avoid actually heating the chips instead of cooling them.

In addition to providing cooling, both have some circuitry to protect against electrical noise in the power line as well as an on/off switch which you can safely use to turn off your entire system without using the Apple's own on/off switch, and the equivalent of three outlets. Both are quiet, and cause no electrical interference.

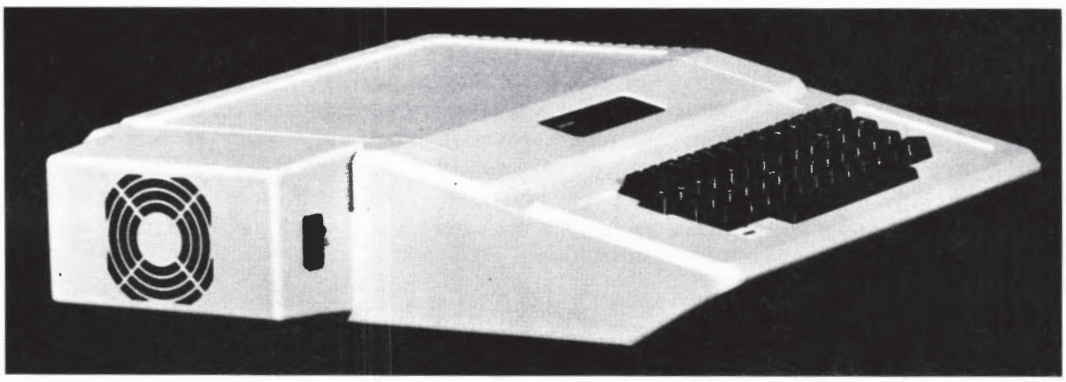


Fig. 12.9 RH Super Fan.

These two fans started out from the same company, but the Super Fan II seems to have gone on with the original engineers while the System Saver went off with the marketers. In the current versions, the Super Fan II (see Figure 12.9) has about twice the air flow of the System Saver, its surge and spike electronic protection ("zener ray option") are far superior, and it is much quieter.

There are some physical problems with this side mounting design. Many Apple owners like to put their Apple under a shelf system which supports their drives over the Apple and the Monitor above the drives. The sides of most of the shelf systems get in the way of the fan. If you must have a side mounting fan, be sure to choose a stand that accommodates it. This warning does not apply to the Monitor III stand sold by Apple with the //e, since these fans fit on this stand after a minor modification, although air flow is not as good. If you've got a newer revision B Apple //e (see Appendix D) sold with the Monitor II and no stand, you have to watch out because the new //e case has wider side walls than the older case so many fans in the stores won't slide on.

If you already have a stand which restricts the sides of the Apple, you can get The Fan from Kemcore Company which fits neatly along the back of the Apple. It has slots and connectors so that it doesn't impair access to the Apple back panel, and it also has surge protection. Another option is to get a stand with a built-in fan from FMJ (Sentry II Cool Stack) or Doss Industries (Apple Center), both of which also permit you to put your Apple under lock and key to protect your peripheral cards.

For well stuffed Apples with five or six cards, the System Saver does not provide sufficient air flow. If you get the M & R Sup 'R Switcher heavy duty external power supply, you'll be starting off with much less heat in the box, under which circumstances a Super Fan II from RH Electronics (approximately 10 cubic feet per minute) should take care of you on up to a full house of eight cards.

Other than that, if you don't mind the noise, you can purchase a very powerful five inch "muffin" fan from almost any electronics distributor (Concord, JDR) for about $10. These fans vary with regards to audible and electrical noise, but at $10 a shot you can afford to try whichever one is most easily available. The fan is placed snuggly against the back of the Apple (be sure to attach a finger guard) and works very well with some of the metal Apple stands such as the Pro-Tech II from Segull Enterprises.

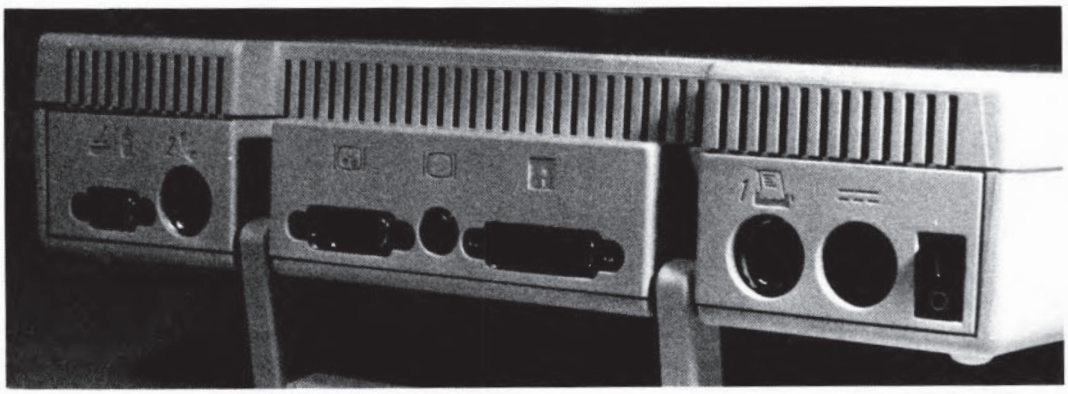
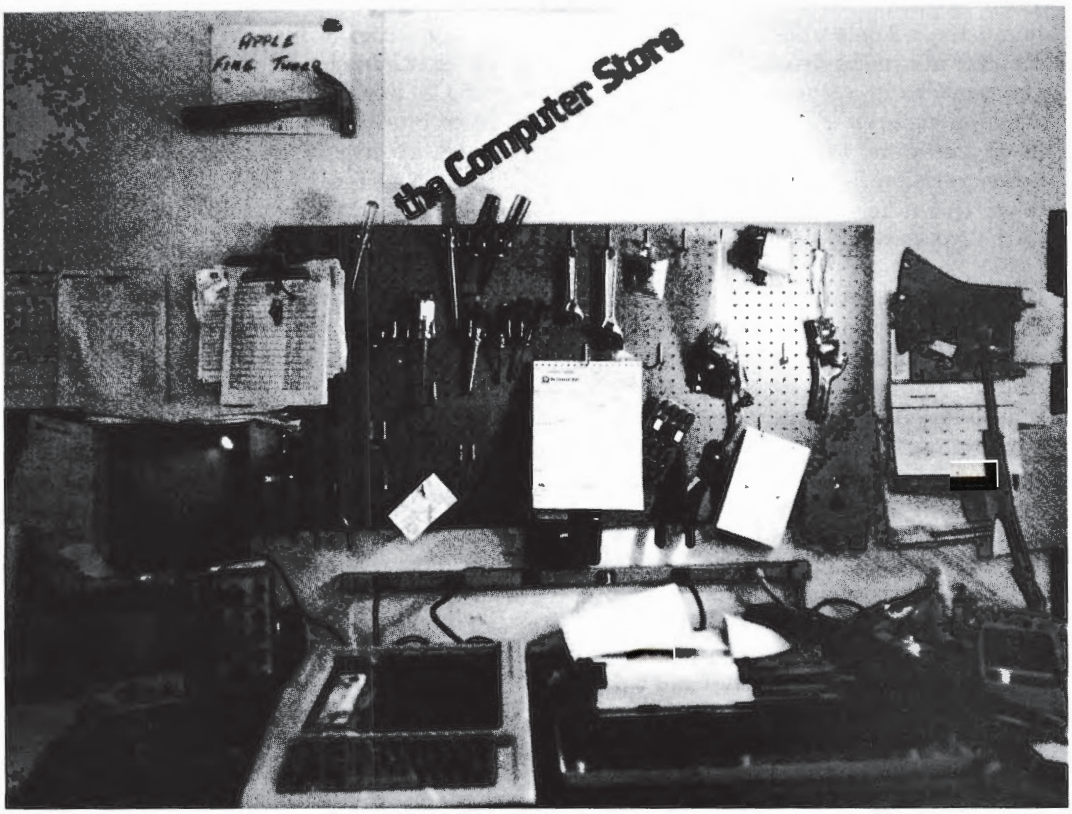


Fig. 12.10 The hi-tech convection cooling system for the //c is based on the handle/stand which assures that air can flow up from beneath the system.


APPLE
FING. TWIGS
the Computer Store

Chapter 13

Electronics and Apple ICs

In electrical equipment, resistors, conductors, capacitors, inductors and switches are grouped together to store and transmit power, while in electronic equipment these same devices are grouped together to store and transmit information. The key to electronic design is the assembly of these fundamental devices into three kinds of special circuits; the reactive circuit, the amplifier, and a newer addition, the logic gate.

Reactive Circuits

As you will recall, a resistor slows down the flow of electrons in a circuit (see Chapter 12). It does this because the substance of the resistor doesn't provide any simple paths for the electrons to follow. Because of the material nature of this effect, a resistor always has the same resistance, no matter what is going on around it electrically.

Reactive circuits are based on a property called "reactance," which is somewhat similar to resistance in that it can hinder the flow of electrons. However, the physical basis of reactance is very different from resistance. It depends on the behavior of electromagnetic fields, and a device can shift rapidly from very high reactance to very high conductance depending on changes in the circuit it is part of. The crucial factor in these shifts is the rate at which voltages are changing in the circuit.

There are two kinds of components that have reactance. One of them has high reactance when the voltages are steady, but shifts gradually into conductance when voltages begin to alternate rapidly. This component is called a "capacitor." The other kind conducts freely while the voltages are steady but acquires greater and greater reactance as the rate of voltage change increases. This second kind of component is called an "inductor."

These reactive components are the flesh and blood of "linear" electronics which includes the design of most radios, televisions, stereos and tape recorders. For the most part, they have been banished to a minor supporting role in the "digital" world of computers, yet there are quite a large number of them scattered throughout the Apple. They serve as the principal means for enforcing time dependence on the behavior of electric currents. In the Apple, they are used in comparatively inaccurate timing circuits in the keyboard and the game connector, in "signal conditioning" circuits in the disk drive read circuitry, and in housekeeping chores throughout the system.

Capacitance

Let's begin with a voltage source and a conducting wire with a resistor in line. Current is flowing in the circuit at a steady rate. If you cut the wire and separate the ends, the circuit is broken. One end of the wire is a potential source of electrons (the "negative" end), and the other end can attract electrons (the "positive" end), but because the ends are separated, no current flows. Now if you take the two cut ends of the wire and bring them very close together but not quite touching, something interesting begins to happen. Under the influence of the attractive end, electrons begin to accumulate in the negative end facing it. Current has begun to flow in the wire although the circuit is still broken.

The electrons do not continue to accumulate indefinitely, however. Electrons repel each other, and as the attractive force causes them to get packed more and more densely into the negative end of the wire this repulsion becomes an important force in its own right. Eventually the repelling force from the densely packed electrons which is inhibiting the arrival of new electrons becomes equal to the attractive force. At this point, current flow stops. The end of the wire has exhausted its capacity to hold electrons under those particular conditions.

Now, if you sharply increases the applied voltage, current flow will start up again until the repulsion again balances the applied force. Whenever the voltage increases, electrons will flow in, whenever voltage decreases, they will be driven out. However, so long as the voltage remains steady, a steady state will be achieved and current flow will halt. Thus when the voltage is steady, the capacitor acts like the broken, open circuit it really is. However, because of the effects of the transmitted electro-magnetic field, while the voltage is changing there will be current flow in this open circuit. This is the way in which the reactance of a capacitor varies.

The measure of capacitance (farads) has to do with how many electrons can be poured into a capacitor before it will reach a steady state at a given voltage. If the two ends of our wire are attached to large flat plates, with the two plates perfectly parallel, very close but not touching, then the same force can pull in a great many more electrons before the density gets so great that repulsion stops the flow. This arrangement has a greater capacity for electrons. If the plates are then brought a little bit closer together, the attractive force will increase and more electrons will flow in.

Capacitors and Time Constants

Devices which specialize in this little maneuver are called capacitors. They are usually made by taking two large flat metal sheets, placing a very thin layer of insulator between them, and rolling up the whole thing into a neat little package. The capacitance of one of these devices is determined by the surface area of its plates and how close together they are. When one of these is placed in our simple circuit and the voltage is turned on, the current will carry electrons into it until its capacity is reached and then the flow will stop.

The interesting question here is: How long will it take for the capacitor to get full? This is determined by the amount of current. The current is proportional to the voltage and to the value of the resistor we put in the circuit ($I = V/R$). If the current passes 10,000 electrons per second and our capacitor can hold 100,000 electrons, then it will take 10 seconds. If we switch in a capacitor which can only hold 100 electrons, the time drops to a tenth of a second. If we now put in a very stiff resistor which sharply reduces the rate of current flow, then the filling time will increase again. In this fashion, by adjusting the value of resistors and capacitors, a characteristic response time or "time constant" can be built into a circuit.

Stray Capacitance

There is a certain scourge associated with all this. The problem comes about when two otherwise unrelated wires happen to pass near each other in a complex circuit. An unintentional capacitance develops, and every time a voltage is turned on in one of the wires some of the electrons will get drawn off into this accidental capacitor until it is full, at which time the circuit will begin to behave normally. The bigger the accidental capacitance, the longer it will take for the circuit to get itself in order. In devices intended to change voltages over periods of a few billionths of a second, stray capacitance can become an enormous hindrance to good performance.

A similar problem slows the maximum speed at which a high speed electronic switch can be thrown. A certain amount of time will always be taken up in filling or emptying the tiny little capacitive pools and puddles that creep into any design. Yet another problem from stray capacitance is the inadvertent coupling together of two wires which are not very close but one of which is carrying a high speed, high voltage spike (see Chapter 12). Capacitive coupling will let the high voltage spike cross into a new circuit even though there is no actual connection between them.

Decoupling Capacitors

There is one problem in digital circuitry for which this sort of property can be very helpful. All digital components effectively sit between a supply voltage and a ground. As they do their work, they permit very brief pulses of the supply voltage to pass through into what may be thought of as the "ground plane" which they share with their neighboring components.

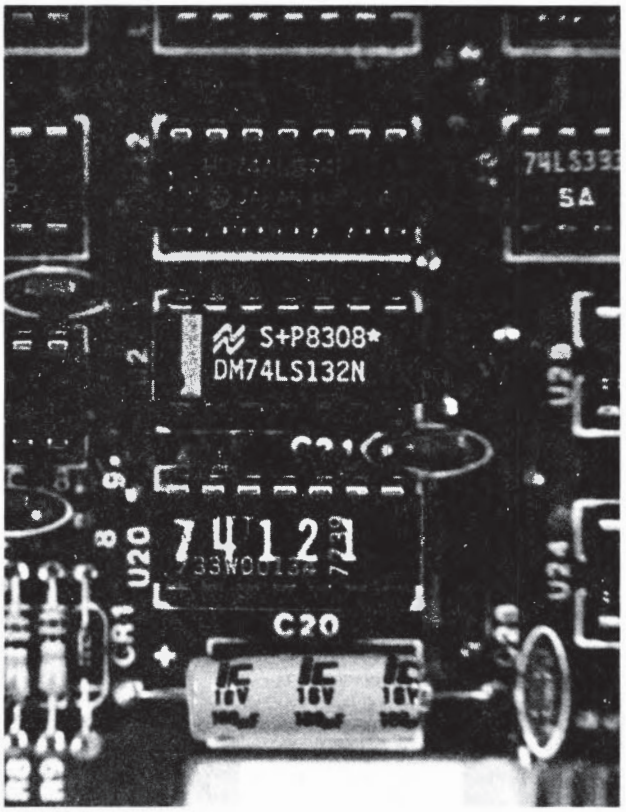


Fig. 13.1 A small capacitor is placed near each digital chip on this card, and a large 100 microFarad decoupling capacitor prevents digital noise from escaping out onto the motherboard power plane.

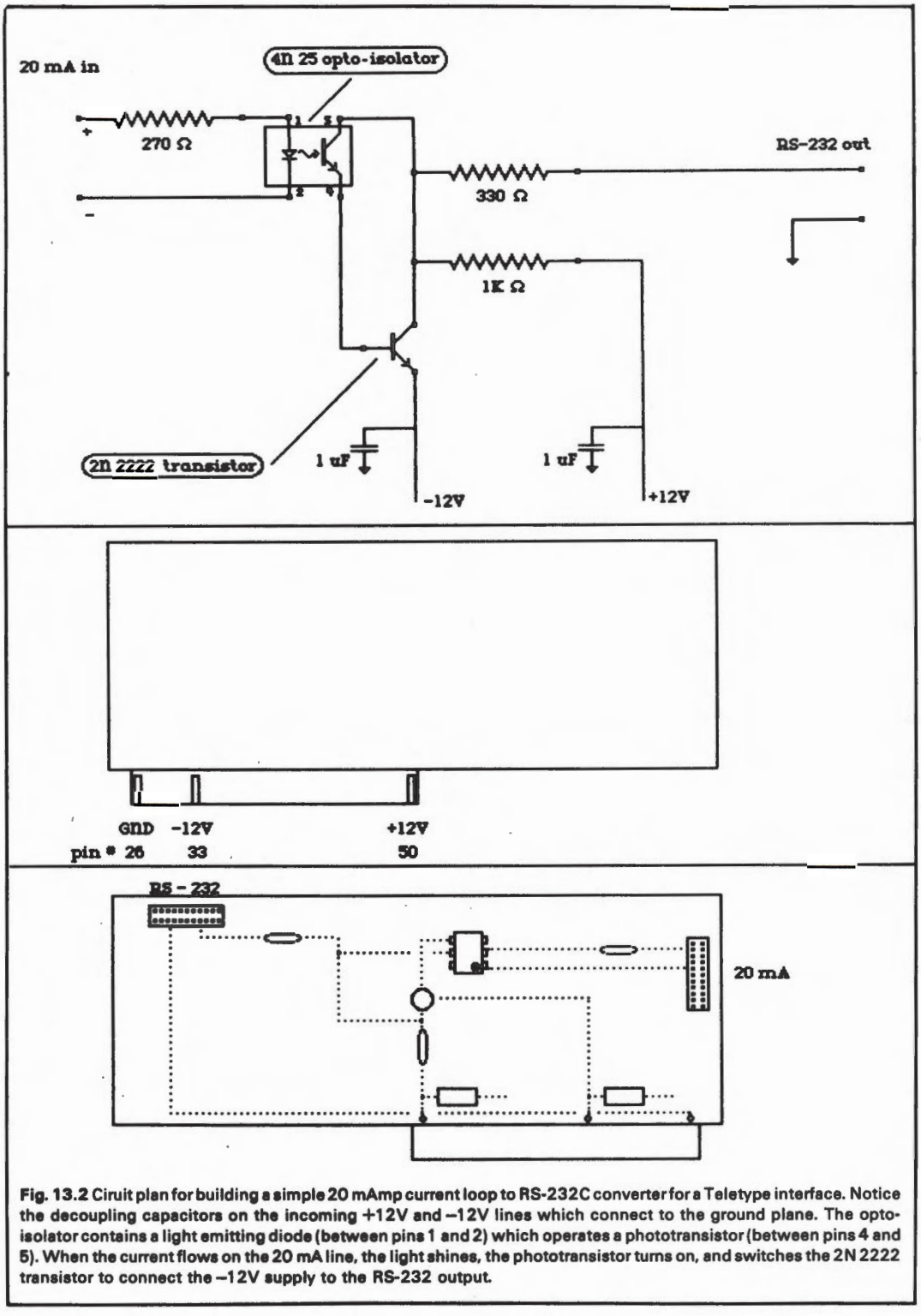


Fig. 13.2 Ciruit plan for building a simple 20 mAmp current loop to RS-232C converter for a Teletype interface. Notice the decoupling capacitors on the incoming +12V and –12V lines which connect to the ground plane. The opto-isolator contains a light emitting diode (between pins 1 and 2) which operates a phototransistor (between pins 4 and 5). When the current flows on the 20 mA line, the light shines, the phototransistor turns on, and switches the 2N 2222 transistor to connect the –12V supply to the RS-232 output.

So as a large digital circuit, such as the Apple, goes about its work, it is constantly dumping millions of tiny high speed voltage spikes into the ground plane. If the ground plane is large, and all the components widely separated, then the little spikes will dissipate their electrons into the surrounding void. In the real world, ground planes are thin and components are densely packed, so all of the errant electrons can start popping their way back up into neighboring chips, causing general havoc and confusion.

To avoid this, you will notice that all digital circuit boards are studded with little capacitors that join the main supply voltage lines to the ground plane (see Figure 13.1). As long as the computer's steady DC supply stays at a fixed level above the 0 volts of ground, the capacitors will keep the power plane separated from the ground plane. However, whenever a little spike zips by, it will present itself to the capacitor as a sharp rapid rise in voltage, and the capacitor will effectively soak up the pulse (see Figure 13.2). Thus, lots of "decoupling capacitors" lead to a nice healthy quiet ground plane.

Inductance

The property of inductance is an even more bizarre result of the behavior of electromagnetic fields. Whenever a current flows in a wire, it creates a magnetic field around the wire in addition to creating an electric field. The lines of attraction in the electric field run perpendicularly outward from the wire, while the lines of attraction in the magnetic field run in circles around the wire.

When a circuit is open and no current is flowing, there is of course no electromagnetic field generated by the wire. However, when the current begins to flow, part of the work to be done goes into driving the force fields out into the space around the wire. These force fields have a feel to them which is not unlike the inertia of a heavy mass when you first try to get it moving. Most of the energy goes into setting up the field, but it requires little additional energy to stay in place. However, when you then throw a switch to break open the circuit, the inertia of the field shows itself again in that it collapses back into the wire a bit sluggishly. Further, as it collapses it pours its energy into the wire, thus driving a collapse current.

The result of all this is that when you first turn on a circuit, the inductive elements soak up some of the energy in the first few electrons, thus acting a bit like a resistor. The faster you try to make the transition from zero to maximum, the more the inductance soaks up energy. When you apply a high frequency change to an inductive element, no current gets through at first, much as if the circuit was broken. Once you reach a steady voltage, however, the inductive effects seem to disappear. Later, when you try to turn the circuit off, current keeps flowing as the collapsing fields dump their stored energy back into the system.

When inductance is desirable, you can arrange a wire into a shape which will create the densest possible magnetic field. This is done by coiling the wire. As the field increases in magnitude, the lines of force become extremely dense in the center of the coil. This makes it even more difficult for the field to build rapidly. The effect is as if the magnetic field was creating an opposing voltage in the wire.

Information and Reactance

From the point of view of digital computer electronics, it is difficult to see how capacitors and inductors can be considered informational circuit elements. However, it is from these components that complex frequency coding and decoding systems in FM radios and other frequency

based carriers are constructed. Many older electronics texts discuss only the reactive circuits such as tuners, filters and resonators based on capacitors and inductors when they discuss the transmission of information.

Amplifiers

An amplifier is a device in which a small current flowing in one circuit is given control over a larger current flow in a second circuit. Such an amplifier circuit is built around one of two special kinds of electrical components; the vacuum tube or the transistor. The first ancestor of the amplifying vacuum tube was created by Thomas Edison just before the turn of the century, and the transistor was invented at the Bell Laboratory by William Shockley in 1947.

In a vacuum tube or transistor, the weaker circuit is able to change the component from a non-conducting insulator to a resistor, and on into a very good conductor. For example, a vacuum tube can be included in a circuit which carries huge currents at thousands of volts. When the tube is operated as an insulator no current flows, but when a few volts are applied to its controlling input, it can be converted into a conductor passing huge currents. An increase of a few volts on the controlling line can cause an increase of a few thousand volts in the main output of the tube. This is what is meant by "amplification," a low power circuit controls a separate high power circuit.

Semi-Conductors

In a vacuum tube, the change in conductance results from the ejection of free electrons into an otherwise non-conducting vacuum, the effect that Thomas Edison discovered. In a transistor, the change can result from the ejection of free electrons into an otherwise poorly conducting slice of specially prepared silicon; a substance also referred to as a "semi-conductor."

The detailed explanation of the way crystalline silicon changes from an insulator to a conductor is the realm of a special branch of physics (solid state or semiconductor physics), but it has all the trappings of alchemy; mixing in a few parts of arsenic (really) here, a little boron or gallium there, etc., in a process called "doping." Arsenic, for example, "builds in" a few extra electrons. Mixing in some gallium or indium, on the other hand, leaves some holes in the crystal matrix of the silicon.

In arsenic doped silicon, the extra electrons serve as negatively charged current carriers, just as they do in copper. Silicon prepared in this way is called "n-type" silicon because of its negative carriers. If you add gallium instead of arsenic, however, you actually produce a slight shortage of electrons. It is convenient to think of this shortage as a scattering of electron-less "holes." These holes can actually move within the silicon, behaving as if they were positively charged current carriers; this is therefore called "p-type" silicon.

Transistors

In a transistor, some p-type silicon can be sandwiched between two pieces of n-type silicon. If you dump extra electrons into the center piece of p-type silicon, its holes will get filled and you will have simulated a solid piece of n-type silicon. In your simulated solid piece of n-type silicon there will be electrons available for current flow throughout the device, so you will have set up current flow in a conductor. If you withdraw your electrons from the center slice, the device will cease to conduct. This is the feat which was accomplished by Shockley in 1947.

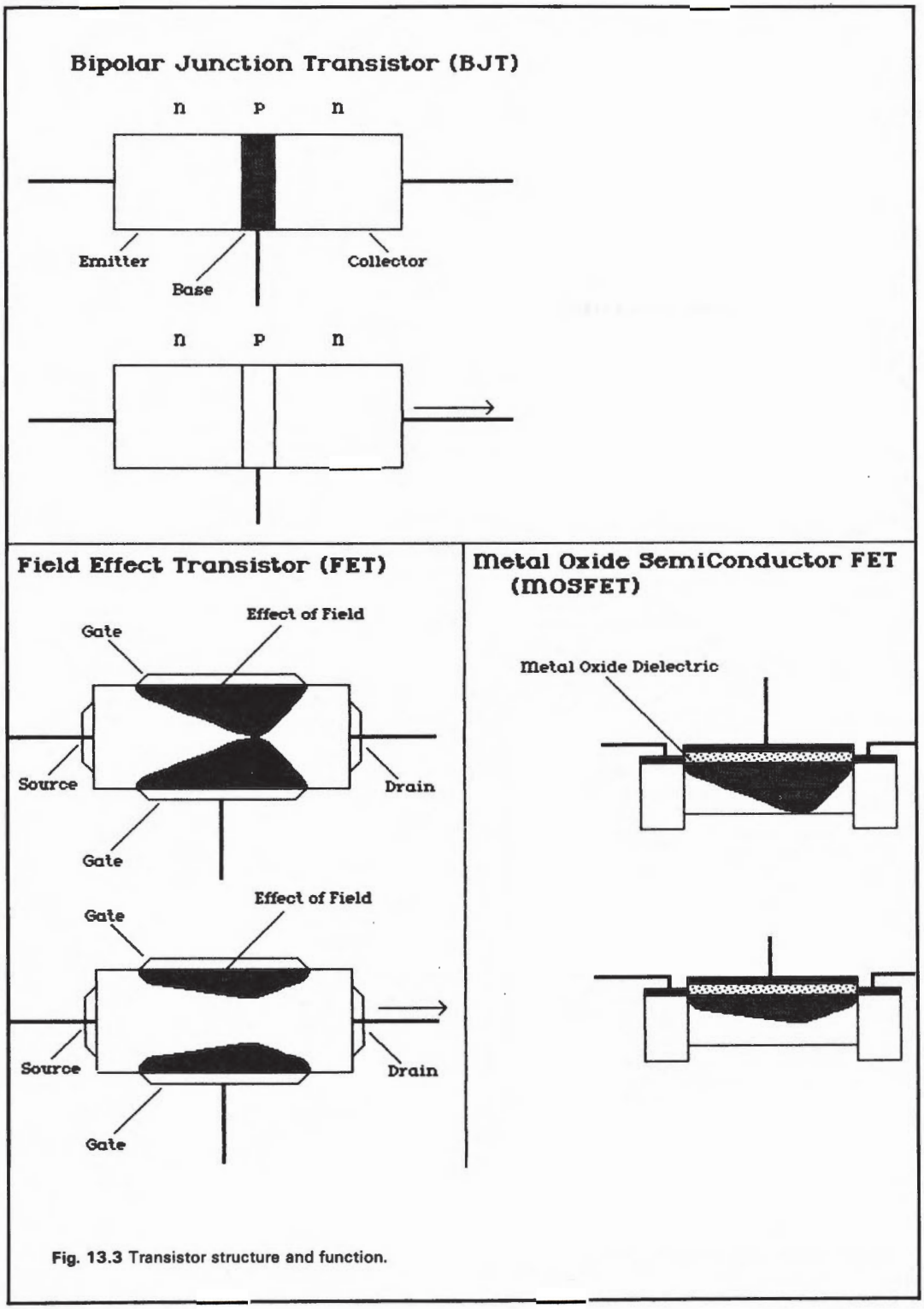


Fig. 13.3 Transistor structure and function.

A transistor performs exactly the same function as a vacuum tube, but vacuum tubes are comparatively large, hot and expensive. This is why the 35 years since Shockley's success have seen the steady replacement of vacuum tubes by transistors in most electronic equipment. Vacuum tubes are still used for high power circuits and for a few other special functions which transistors cannot handle, but they have virtually disappeared from standard electronic equipment.

Two Kinds of Transistors

Transistors come in two large families. For years, there has been a sort of truce in which the older kind has been used in most of the simpler chips, but the newer type has been used in the very high density chips such as microprocessors. More recently, a bitter competition has erupted in consumer electronics for domination of the market for simpler chips.

The kind of transistor described a few paragraphs earlier is called a Bipolar Junction Transistor (BJT). These transistors always have at least three parts; two n-regions with a p-region in the middle, or two p-regions with an n-region in the middle (see Figure 13.3).

The other grand class of transistors is called Field Effect Transistors (FETs). The most important of the FETs is a kind in which there is only one thin strip of n-region silicon. Normally, it exists in the circuit as a conductor. However, a small patch of metal oxide is laid down over the middle of the strip to insulate it from a conducting plate above. This plate can be charged up by its control line. When the plate fills with electrons, it generates an electric field over the n-region silicon below it, forcing the free electrons out of that area of the silicon. When this happens, current flow in the silicon stops.

This arrangement is called a Metal Oxide Semiconductor Field Effect Transistor (MOSFET). Because these transistors have so few parts, they are the easiest to manufacture in extremely high density designs. The 6502 is made from this kind of n-region transistor, so it is called an "NMOS" microprocessor. This is the source of the name of the 6502's original manufacturer MOS Technology.

Electronic Circuits Built from Transistors

In a standard amplifier, the controlling current is varied gradually to produce a gradually varying large output. This is how a weak radio signal is used to control your powerful speakers. Transistors don't do too well in this sort of task because their output doesn't follow their input in a sufficiently simple way. As a result, solid state amplifier circuits are built out of groups of transistors which work together to simulate an old vacuum tube. The new device is called an Operational Amplifier (Op Amp).

The Apple //c uses two fancy op amps to control and isolate (see Chapter 14) the audio and video outputs. One is included in the AUD (see Chapter 8, Figure 8.3c) package, and the other is in the VID (see Chapter 5, Figure 5.8c) package. These devices are called "hybrid amplifiers" because they use both digital and analog circuitry, all wrapped up in a single package. The II, II+ and //e have a simple LM 741 op amp for managing the input from a cassette tape recorder, but the //c has no cassette interface so the LM 741 is gone. Finally, the //c has an LM 311 op amp (see Chapter 8, Figure 8.3c) which maintains a constant surveillance on the incoming DC power from the external power supply. If it detects a drop below a certain minimum, it activates a 555 timer which causes the power light on the keyboard to flash.

Transistors do excel in the extreme function of the old amplifier tubes, which is to turn a circuit completely on or completely off, with no variation in between. The fastest modern transistors can do this in less than a billionth of a second (an electron traveling at the speed of light cannot make it from one end of the Apple's motherboard to the other in a billionth of a second, so these very fast transistors are used effectively only in physically small computing machines).

Although transistors were born as small, inexpensive amplifiers, their availability has led to the emergence of a third kind of basic electronic component. In addition to resonant circuits and amplifiers, electronic design now depends on small circuits called "logic gates."

Logic Gates

A logic gate is built from five or 10 transistors and a few resistors. There are three fundamental kinds of logic gates; the AND gate, the OR gate, and the INVERT gate. An AND gate has two inputs and one output. If, and only if, BOTH of its input lines are receiving "on" signals will it turn on its output.

An OR gate also has two inputs and one output, but it behaves a little differently. If one of its inputs, or the other input or both of its inputs are "on," then it turns on its output. The INVERT gate is even simpler. It has just one input and one output. As its name suggests, if it receives an "on" signal, it puts out an "off" signal. If, on the other hand, it receives an "off" signal on its input, it puts out an "on" signal.

A 19th century mathematician named Boole worked out a complete and elaborate form of algebra based on combinations of logical operations. In this "Boolean" algebra, he used only ones and zeros, but was able to perform any sort of calculation. Electronic designers who began working with logic gates in the early 1960s heartily embraced Boolean algebra because it showed how to hook together AND, OR and INVERT into increasingly elaborate circuits with increasingly elaborate algebraic behavior.

The three basic logic gates can be hooked together to provide a few more basic logic functions including "NAND" (if both inputs are positive then put out a negative), "NOR" and "XOR" (if one input is on or if the other input is on, but not if both are on, put out an "on").

The Integrated Circuit

Early on in the development of the logic gate as a fundamental electronic device, they were made by wiring together the necessary discrete transistors and resistors. But in the early 1960s, Jack Kilby of Texas Instruments invented a truly revolutionary way of making the gates. The entire logic gate circuit was printed by photolithography on a single piece of silicon and sold as an Integrated Circuit (IC).

Before Kilby's invention, you bought transistors and resistors and fiddled to make each individual logic gate work, soldering together the components one by one. But the advent of the integrated circuit truly transformed the task of the electrical engineer. A new kind of design was possible that had its roots more in Boolean algebra than in electrical theory. A "logic designer" purchased complete "logic circuits" and connected them together according to Boolean dictates, and not necessarily needing to think about resistance, capacitance, volts or amps. An AND gate was designed and debugged once, cast in silicon, and mass produced. Wholesale and daily reinvention of this wheel halted abruptly.

During the course of the 1960s, two lines of advance were taking place parallel to each other. Logic designers worked at the next level of complexity to assemble groups of logic gates into circuits which performed a variety of mathematical operations and memory functions. At the same time, specialists in the fabrication of IC silicon chips were learning how to put increasing numbers of gates into a single IC in a Dual In-line Package (DIP). By the late 1960s designers could design very sophisticated devices by adding together chips with fairly elaborate collections of logic functions. It was in this environment that the first microprocessor was designed.

Apple II Electronics

Tables 13.1, 2 and 3 have lists of the discrete chips used in the construction of the Apple II/ II+, //e and //c. The makeup of the II is fairly typical of standard electronic design today. There is one very high density NMOS microprocessor, some fairly high density memory chips, and the remainder are all integrated circuits of rather modest complexity.

The //e and //c are very unusual in their electronic constitution. Apple expected to make a huge number of systems with the identical set of logic functions, so an engineer (Walter Broedner) who came to Apple from Synertek designed the original versions of two specially made chips of fairly high density (the Memory Management Unit, see Chapters 21, 25 and 26; and the Input/Output Unit, see Chapters 5 and 9) which are used in the //e and //c to provide all of the functions of dozens of smaller chips.

In the //c, the MMU and IOU have been redesigned a little bit to accommodate special features of the //c. The MMU loses its DMA line (see Chapter 27) and the R/W 245 lines which is used in the //e to connect the slots to the main data bus (see Appendix A). In their place, it takes over control of some of the $C0 page hardware select functions which are managed by discrete chips in the //e. The //c's IOU loses its annunciator outputs (see Chapters 9 and 14), but gains a set of switches, inputs and gates it can use to monitor mouse movements and generate interrupts with its new IRQ line.

The //c and //e both have a Programmed Array Logic (PAL) chip, which is called TMG (for "timing") in the //c. This chip is effectively identical in the two machines, and is used to accept the master 14 mHz clock as an input and then generate a number of special timing outputs.

The //c has two more special chips. One is the GLU, which is a PAL device whose modest task is to replace three or four simple logic chips and to take over one or two functions that could no longer be squeezed into the IOU (see Chapters 22 and 26). The other is a custom made device called the Integrated Woz Machine (IWM; see Chapter 23). It replaces all of the circuitry of the old Disk II interface card. This exact same chip is also used as the disk controller in the Macintosh.

Finally, in addition to all the specially made chips, the 65C02, and all the standard chips listed in Table 3, the Apple //c has four chips used for serial RS-232C communication. There are two large 6551 Asynchronous Communications Interface Adapters (ACIAs; which are described in detail in Chapter 16) and there are two chips which handle some mundane aspects of the RS-232C electrical interface.

One is an MC 1488 which acts to accept TTL level voltages (0 to +5) used inside the Apple (see below) and convert them to the higher voltages (-12 and +12) used in RS-232C communication (see Chapter 16). There is also an MC 1489 which performs the necessary opposite function of converting incoming RS-232C voltages into TTL voltages. The 1488 and 1489 are

not voltage converters of the sort described in Chapter 12, rather the 1488 has -12 and +12 voltage lines coming in, but uses a TTL voltage to operate the switch that can connect the higher voltages to the chip's output. The 1489 works in a similiar way.

Chips on Apple II/II+ Motherboard

74LS	00	(1) Quadruple two-input positive NAND gate
74LS	02	(3) Quadruple two-input positive NOR gate
74LS	04	(1) Hex Inverter
74LS	08	(2) Quadruple two-input positive AND gate
74LS	11	(1) Triple thee-input positive AND gate
74LS	20	(1) Dual four-input positive NAND gate
74LS	32	(1) Quadruple two-input positive-OR gate
74LS	51	(1) AND-OR invert gate
74LS	74	(3) Dual D-type positive edge triggered flip flop w/preset and clear
74LS	86	(1) Quadruple two-input exclusive-OR gate
74LS	138	(4) Three to eight lines decoder/demultiplexer
74LS	139	(2) Dual two to four line decoder/demultiplexer
74LS	151	(1) One of eight selector/multiplexer
74LS	153	(4) Dual four line to one line data selector/multiplexer
74LS	161	(4) Synchronous four bit counters, binary, direct clear
74LS	166	(1) Eight bit shift register
74LS	174	(3) Hex D-type flip-flop, single rail output, common direct clear
74LS	175	(1) Quad D-type flip-flop, complementary outputs, common direct clear
74LS	194	(2) Four bit bidirectional universal shift register
74LS	195	(1) Four bit parallel access shift register
74LS	251	(1) Data selector/multiplexer, true and inverted three-state output
74LS	257	(5) Quad data selector/multiplexer, noninverted three-state output
74LS	259	(1) Eight bit addressable latch
74LS	283	(1) Four bit binary full adder
LM	741	(1) Op-Amp
NE	558	(1) Quad Timer
NE	555	(2) Timer
8T	28	(2) Quad transceiver
8T	97	(3) Hex tri-state driver
	8304	(1) Octal Transceiver (Rev. 7 boards)
	2316	(1) Video Character Output ROM
	2316	(6) 2K x 8 ROM (Monitor and Interpreter)
	4116	(24) 16K x 1 dynamic RAM
8Y	6502A	(1) Microprocessor

Chips on Apple IIe Motherboard

74LS	02	(1) Quadruple two-input positive NOR gate
74LS	10	(1) Triple three-input positive NAND gate
74LS	109	(1) Dual JK positive edge triggered flip-flop with preset and clear
74LS	125	(1) Quadruple bus buffer gate with three state outputs
74LS	138	(1) Three to eight line decoder/demultiplexer
74LS	154	(1) Four line to sixteen line decoder/demultiplexer
74LS	166	(1) 8 bit shift register - parallel/serial input, serial output
74LS	244	(2) Octal buffer/line driver/line receiver, non-inverted three state output
74LS	245	(1) Octal bus transceiver - noninverted three state outputs
74LS	251	(1) Data selector/multiplexer, true and inverted three state outputs
74LS	374	(1) Octal D-type flip-flop, three state outputs, common output control, common enable
LM	741	(1) Op-Amp
NE	556	(1) Quad Timer
	2316	(1) 2K x 8 ROM (Keyboard Character Codes)
	2332	(1) Video Character Output ROM
	2364	(2) 8K x 8 ROM (80 column firmware, diagnotics, Applesoft, and Monitor
	6664	(8) 64K x 1 RAM
AY	3600	(1) Keyboard decoder
SY	6502B	(1) Microprocessor
IOU-IIe		(1) Input Output Unit
MMU-IIe		(1) Memory Management Unit
PAL		(1) Programmed Array Logic timing generator

Chips on Apple IIc Motherboard

74LS	32	(1) Quadruple two-input positive OR gates
74LS	161	(1) Synchronous four bit counter, binary, direct clear
74LS	166	(1) 8 bit shift register - parallel/serial input, serial output
74LS	245	(1) Octal bus transceiver - noninverted three state outputs
74LS	251	(1) Data selecter/multiplexer, true and inverted three state outputs
74LS	374	(2) Octal D-type flip-flop, three state outputs, common output control, common enable
AUD		(1) Hybrid audio amplifier
VID		(1) Hybrid video amplifier
LM	311	(1) Voltage comparator Op Amp
	7905	(1) Voltage regulator
NE	556	(1) Quad Timer
NE	555	(1) Timer
NEC	2401	(1) Quad optocoupler
MC	1488	(1) Quad RS-232C line driver
MC	1489	(1) Quad RS-232C receiver
	2316	(1) 2K x 8 ROM (Keyboard Character Codes)
	2364	(1) Video Character Generator ROM
	23128	(1) 16K x 8 ROM (Monitor, Applesoft, and I/O Firmware)
	6664	(16) 64K x 1 RAM
AY	3600	(1) Keyboard Decoder
NCR	65C02A	(1) Microprocessor
SY	6551	(2) ACIA (Asynchronous Communications Interface Adapter)
IOU-IIc		(1) Input Output Unit
MMU-IIc		(1) Memory Management Unit
GLU		(1) General Logic Unit (PAL)
TMG		(1) Timing Generator (PAL)
IWM		(1) Integrated Woz Machine (Disk IIc Interface and Controller)

Standard ICs in the Apple II Family

Custom ICs such as the MMU, IOU and IWM are only economical in very large quantities, so most electronic devices made today still use the now classic set of discrete logic chips. Most of these cost only 20 or 30 cents, but they do suck up power, create heat, introduce quality control problems and take up a lot of space on a circuit board.

One clear token of the success of the Apple II family of computers, therefore, is to notice how many standard discrete logic chips are used in the system. In an Apple II or II+, there are 46 of these on the motherboard, five on the disk controller, and you would need eight or nine more to provide two serial ports. This comes to a total of about 60. The comparable figure for a similarly equipped Apple //e including two on the extended 80 column board is 28 discrete chips. All of these functions are accomplished in a //c with just seven discrete logic chips.

The discrete logic IC chips used in the various members of the Apple II family fall into a few standard categories. The simplest chips provide unadorned logic gate functions. More complex chips provide memory, data manipulation and mathematical functions.

Simple Gates

Most modern logic DIP packages contain several completely separate logic gates. They are together merely for economy and have no functional connection. A good example is the chip numbered "74LS 02," which is used in both the II and the //e. The formal name of this IC as listed in the Texas Instruments Data Book is a "Quadruple two-input positive NOR gate" which means that there are four simple NOR gates in the package. The number "02" signifies that it was one of the first products designed in the "74" series sometime in the 1960s. This "74" series of chips from TI has been the brick and mortar of which most of modern electronics has been built. Recent products in this line have designations like "74LS 670" and are far more complex.

Latches, Flip-flops and Registers

Several of the 74-series chips provide simple data storage functions. The latch is the simplest element in this series. It is made from just two NOR gates connected in a figure eight and can hold either a one or a zero. A master-slave flip-flop or an edge-triggered flip-flop is built of a few of these latches and provides a variety of means of manipulating the one bit being stored. A register is usually a package containing several connected flip-flops. In a parallel to serial shift register such as the 74LS 166 used in the video system of the II, II+, //e and //c, a full parallel eight bit byte is loaded into the eight cells of the register, but, on command, the contents of each register is passed down the line. The ones and zeros are then read off the end of the register in serial form. This is how the Apple converts a byte of video data from the character generator ROM (see Chapter 5) or from graphics display memory into a series of on and off signals to control the electron gun in the CRT.

Decoders and Multiplexers

Decoding chips are used to translate incoming addresses into signals to individual chips. One type of simple decoder has four output lines, but only two inputs. The two inputs carry in a binary number which will be equal to 0, 1, 2, or 3 (00, 01, 10, 11). The decoder responds to the binary number by turning on the appropriate one of its four outputs.

A multiplexer is used to select between two different sources of information. A common design has four outputs but eight inputs (two sets of four inputs). In response to an external signal it connects four of its inputs to the four outputs, leaving the other four inputs unused.

Counters and Adders

Simple low level math can be relegated to stand alone ICs. Counting and adding chips are very popular in cards which need to do a little bit of processing of incoming data before handing the information over to a microcomputer system. In the Apple, the video display generator does its work without help from the 6502, so in the II and II+ it uses a 74LS 283 to do a little math for it.

Families of Logic Chips: TTL and CMOS

Logic designers do live something of a charmed existence, free from worry of fiddling directly with transistors, but they are forced to deal with some special electronic properties of the logic chips. The most important considerations are heat and speed. Often, the design possibilities are constrained by the actual speed at which the transistors inside the chips are doing their switching. Similarly, the energy consumed by many tens of thousands of transistors results in heat which limits their density and creates demands on the power supply.

Since logic gates were first built into integrated circuits, there has been a continual attempt to improve their performance by somehow increasing their speed and/or decreasing their consumption of power. These factors are determined by the particular way in which the various transistors and resistors are arranged in the fundamental logic gate building blocks.

In the 74 series from Texas Instruments, the arrangement is called Transistor Transistor Logic (TTL). These gates use the BJT type of transistor and always require +5 volts to be available for generation and reception of signals. TTL chips come in a variety of flavors which reflect both the needs of special designs and also general progress in semiconductor technology.

The 74LS series

The original 74 series logic gates had a delay time of nine nanoseconds (billionths of a second) and each consumed 10 milliwatts of power continuously. In a later "low power" version, only one milliwatt was consumed, but the latency time went up to 33 nanoseconds. The most important innovation however was the inclusion of a "schotky-type" transistor in the gate. Without explaining exactly what this is, suffice it to say that this produced power consumption of two milliwatts and a delay of 9.5 nanoseconds. These gates are designated 74LS for Low power Schotky. They are currently the dominant circuit component type in the world of digital electronics. The newest 74F series chips have a delay of just three nanoseconds and a power consumption of four milliwatts, but these are not yet in widespread use.

The 4000 series from Motorola

An increasingly important alternative to 74LS TTL chips is an entire family which duplicates all the logic functions, but is based on a kind of MOSFET transistor. The transistors in these logic gates are arranged in complementary pMOS and nMOS pairs, so the logic family is formally called Complementary Metal Oxide Semiconductor Field Effect Transistor and is universally abbreviated CMOS. These chips are usually given numbers which are in the range of 4000 to 4999, but the last three digits are always the same as the 74LS chip with the same function. A 74LS 166 becomes a 4166. Texas Instruments has recently begun producing its own CMOS chips and these are given the designation "74C."

The two principal design differences between CMOS and TTL are that CMOS chips consume virtually no power at all when they are not actually in use and that CMOS chips will operate properly with any supply voltage from three volts to 18 volts. These two factors together make CMOS absolutely ideal for battery operated portable equipment.

There are, however, two serious drawbacks to the use of CMOS logic. The first concerns reliable manufacture, assembly and use; they are very sensitive to static charges and are easily destroyed. The second has been more damning; they typically have delay times in the range of 50 or 60 nanoseconds. These chips would not function in some parts of the Apple's video generator for instance. Further, as they approach continuous operating speeds of one or two megahertz, their power consumption becomes greater than the 74LS chips they are competing with.

An enormous research and development effort has been poured into the CMOS speed problem. Substantial improvements have been achieved in the lab, so CMOS promises to become even more important over the next few years. The 65C02 used in the //c has CMOS circuitry internally, but it is able to interface conveniently with the TTL type voltages used for the rest of the Apple //c's chips.

Building and Testing Boards for the Apple Bus

Although circuit design and construction is not for the beginner, you don't need to be an electrical engineer. There are innumerable situations in which laboratory scientists, film technicians, or managers of small plants may need to set up a simple interface that just doesn't seem to be available anywhere (see Figure 13.2). At the simplest level, you can use the Apple bus as a five and 12 volt power supply with no ground loop problems. If you are an electrical engineer, the Apple bus can be great fun to work on for all sorts of remarkable projects. The full bus specifications from Apple are provided in Appendices B and C.

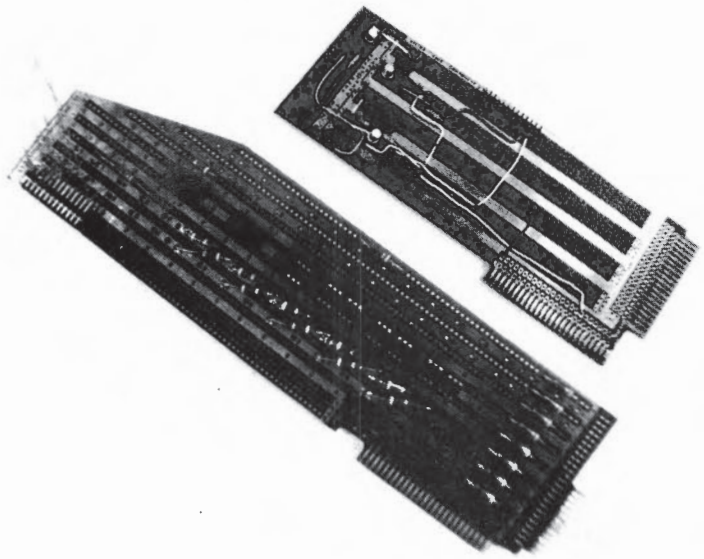


Fig. 13.4 Vector prototyping board (top) and Hollywood Hardware Pro-1 (bottom).

Prototyping Boards

In either case there are a few products available which can help. The first thing you need of course is a "prototyping board," and there are a variety of these available. If you're near an electronics supplier, the easiest thing to get your hands on is usually the 4609 Plugboard from Vector Electronic (see Figure 13.4). This board is similar in size to the Hobby/Prototyping Board from Apple Computer. Of the two, the Vector board is a little easier to use for quick and dirty projects, but the Apple board is of better quality.

If you're going to order a board, the best available is the PRO-1 from Hollywood Hardware (see Figure 13.4). It accepts up to 52 of the 16 pin sockets, is numbered and lettered by row, and has all supply voltages clearly labled in several locations around the board. Douglas Electronics has a general purpose breadboard and a wire-wrap panel, both of which are larger than the Vector board but not as large as the PRO-1.

Testing Equipment

To test activity on the bus, you can work from an "extender board" which makes the signal pins accessible several inches above the motherboard connector. You can get an extender board from Douglas Electronics, California Computer Systems, or SSM.

For full scale testing of professional design work, you may want to consider the Bus Rider from RC Electronics. This card is a full scale logic analyzer designed for the Apple II bus and can be triggered on any specified address or data, on IRQ or R/W, or from any one of four externally supplied signal inputs. It has four 512 by eight buffers and can acquire 512 samples at one MHz.

Recommended Reading in Electricity and Electronics:

BEGINNER

Electricity—Edited by Harry Mileaf, Hayden Book Company Inc., Rochelle Park, New Jersey.

This book is available as seven separate sections or bound together as a single volume. It has more illustrations than text and assumes absolutely no previous knowledge of electricity. Sections on test equipment and electric motors will be irrelevant for those interested in computer electronics.

Electronics—Edited by Harry Mileaf, Hayden Book Company Inc., Rochelle Park, New Jersey.

The section in this book on semiconductor devices is fairly good, but the book is strongly oriented towards radio electronics.

Understanding Solid-State Electronics—By Texas Instruments Learning Center (available at Radio Shack stores).

A well written introduction to the various types of integrated circuits.

Understanding Digital Electronics—By Texas Instruments Learning Center (available at Radio Shack stores).

This book explains how logic circuits are connected together to perform complex functions. The theme of the book is to explain how an electronic calculator works.

INTERMEDIATE

Electronics and Instrumentation for Scientists—by Howard Malmstadt, Christie Enke and Stanley Crouch, Benjamin/Cummings Publishing Company.

Not that everyone interested in electronics is a scientist, but there just aren't too many good intermediate electronics books for businessmen or writers. The principal thing is that this book does not expect any special background in physics or electrical engineering, but provides a very clear account of all aspects of linear and digital electronics, including A/D conversion, and the use of Op Amps.

ADVANCED

The Art of Electronics—By Paul Horowitz and Winfield Hill, Cambridge University Press, New York.

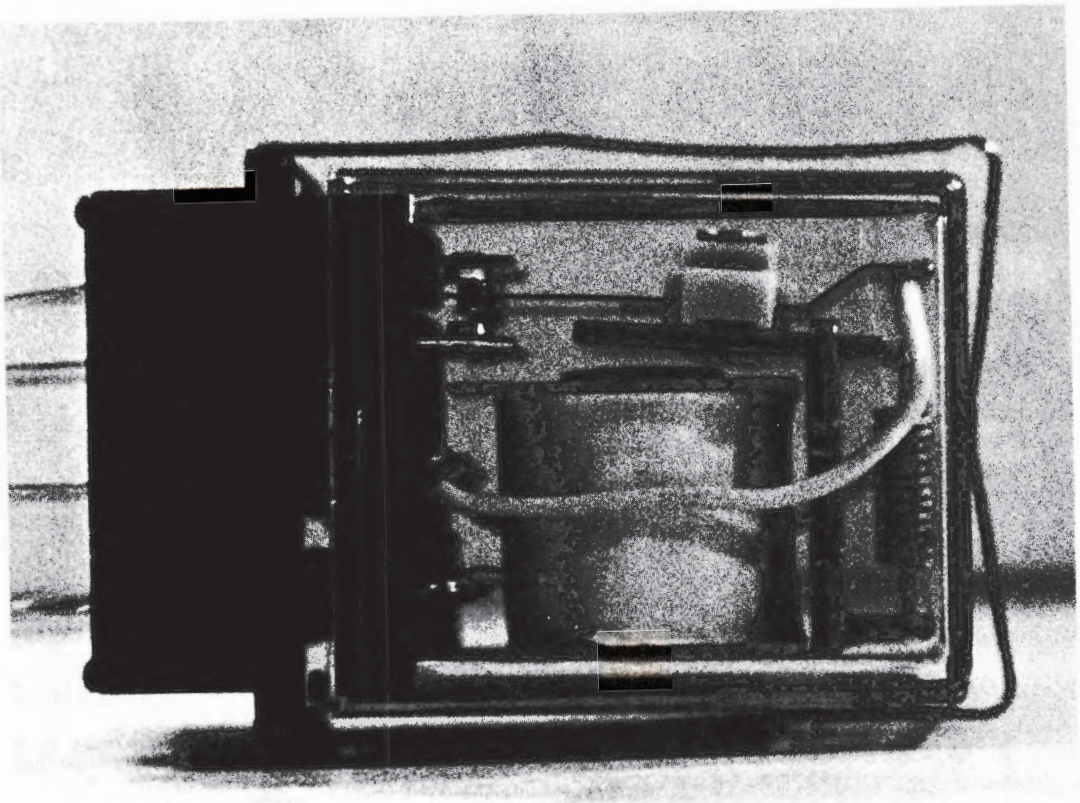
If you're going to read only one book on advanced electronics, this should be it. This Horowitz and Hill book has caused quite a sensation among electronics types. Warning: don't even start if you haven't already got a good grasp of the fundamental concepts.

The TTL Data Book for Design Engineers—available from Texas Instruments, Austin, Texas.

This is the electronics "Bible." It is actually very simple to use and can be very handy to have around even if you're only an advanced beginner. Chips are listed by number and described at several levels of detail.

Handbook of Semiconductor and Bubble Memories—(1982) by Walter A. Triebel and Alfred E. Chu, Prentice Hall, Englewood Cliffs, New Jersey.

Everything you might want to know about static and dynamic RAMs, the various types of ROMs, and the new magnetic bubble memories.



Chapter 14

Switches, Counters and Converters

Calmly, quietly, the Apple interacts with the flow of change in the domains of space, time and energy which swirl around it. A finger touches a keyboard, and the Apple assigns meaning. The 6502 completes a complex calculation and then activates a music synthesizer to generate a tone that has never been heard before. Inside a test tube in a darkened chamber, a biochemical reaction slowly emits photons, a photomultiplier reacts, and the Apple records the instant in time at which each electron released its quantum of light.

The nearly infinite variety of interactions with the physical world in which the Apple can engage are based on switching, counting and analog conversion. These three categories of action are distinguished more by the level of abstraction they signify for the Apple than by any profound difference in the electronics involved. In all cases, the physical phenomena must be represented by numbers which the computer can store and manipulate.

The numerical representation of a switching function is convenient because of its likeness to the digital information system used within the computer. A switch that is on can be represented by a binary 1, while a switch that is off can be considered as a numerical value of 0. The passage of time has an equally obvious numeric description. All that is required is a stream of discrete binary pulses arriving a steady frequency and a counter to describe how many such pulses have arrived. The number of pulses equals the amount of time.

The third class of actions deals with continuously varying signals. These are unlike switches or repetitive pulses in that they cannot be neatly described as either one or zero at any given time. Such "analog" signals are usually described as a series of numbers (eight bit, 12 bit or 16 bit numbers, depending on the precision of the description) each of which is a measure of the amplitude of the signal sampled at a given point in time (see Figure 14.4). The varying signal is thus represented in memory as a series of numeric measurement values.

To acquire the series of numbers you use a cycle of sampling and measurement which is called Analog to Digital Conversion. Similarly, a model of the original waveform can be regenerated by a device called a Digital to Analog Converter, which reads in each of the series of stored numbers and produces appropriate voltage steps.

Various combinations of switches, counters and converters are hidden within familiar input and output devices, such as keyboards and joysticks, as well as in the somewhat less familiar voice output and drawing pad input systems. Those devices are all described in detail in

Chapters 8 through 11, while this chapter is concerned with the switches, counters and converters themselves. Chapter 15 covers laboratory and industrial data acquisition systems which are sold directly on the basis of the speed and power of these three fundamental elements.

Switches

The ideal switch acts like an insulator when it is open (infinite Off-Resistance), perfect conductor when it is closed (zero On-Resistance), and can zip instantaneously from one state to the other. Further, whoever or whatever operates this ideal switch should be completely "isolated" from the electricity in the switched circuit (i.e., you should not get a shock when you flip a light switch). Needless to say, there is no such thing as an ideal switch and you have to trade off among the four properties of Off-Resistance, On-Resistance, Speed and Isolation, all depending on what the switch is supposed to be doing.

Solid State Switches

The kind of solid state transistors that make up most of the Apple's circuitry are among the fastest switches. Unfortunately, although they can throw back and forth several million times a second, their performance in regards to the other three properties is so miserable as to render them almost useless except for interactions with other TTL "logic" circuitry (see Chapter 13).

BJTs

The detailed basis of switching in transistors is explained briefly in Chapter 13. It depends on changing the resistance of a short stretch of semiconducting silicon by injecting electrons or "holes" into it. In the Bipolar Junction Transistors (BJTs) used in most of the Apple's TTL chips, the input line which does the injecting (and therefore controls the "switch") actually becomes part of the circuit it is controlling (see Chapter 13, Figure 13.3). There is therefore effectively no isolation at all between the controlling circuit and the switched circuit.

When a BJT switch is turned on, it can conduct electricity in only one direction, so you have to be very careful if you use them with signals that might involve alternating current. Further, if you apply voltages much in excess of six volts in either direction, many BJTs will go into a "breakdown" mode in which the controlling input loses control of the switched circuit. Most BJTs have enough resistance to start heating up rapidly if you try to run much more than 100 milliamps of current through them, so they must be provided with metal "heat sinks."

The principal reason for using BJT type switches is that they are very easy to interface with the rest of the Apple's circuitry. There are eight circuits on the Apple II, II+ and //e internal Game I/O Connector which are intended for BJT switching. The four "annunciator" lines each provide a five volt output when selected from software, and each of these may be used as the control input for a BJT-type switch outside the Apple. The "strobe output" is similar except that it turns on for just one half of a microsecond when it is addressed by a program. The three "pushbutton" inputs make it possible for a program to read the five volt or zero volt output from an external switched circuit (see Chapter 26). You must have a II, II+ or //e to get full use of these circuits since the annunciator and strobe outputs have been removed from the //c.

In situations which require switching of up to about 60 volts or currents up to about 1.5 amps, you can take advantage of a special kind of high power semiconductor switch. The Texas Instruments SN75000 and ULN 2000 series of peripheral drivers and Darlington Switches

are hundreds of times slower than simple BJTs and aren't useful for alternating current. However, they do greatly extend the range of control tasks for which solid state switches can be used. They are easy to interface with standard TTL circuitry and don't require special physical designs for metal heat sinks.

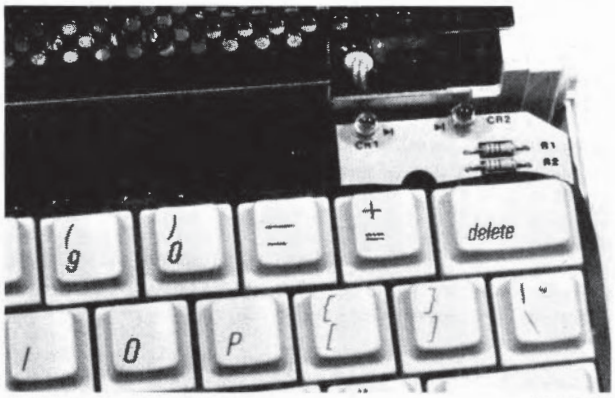


Fig. 14.1a The two LEDs (light emitting diodes) that light up on the //c keyboard are identical in function to the LEDs used inside an optocoupler.

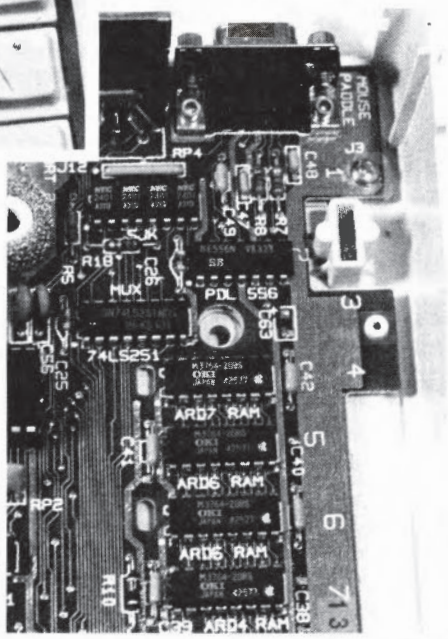


Fig. 14.1b The package marked NEC 2401 contains four LED/photodiode pairs. These couple and isolate inputs from the //c's mouse/game port. The isolation helped make it possible to retrofit the //c to be electronically compatible with the Macintosh mouse.

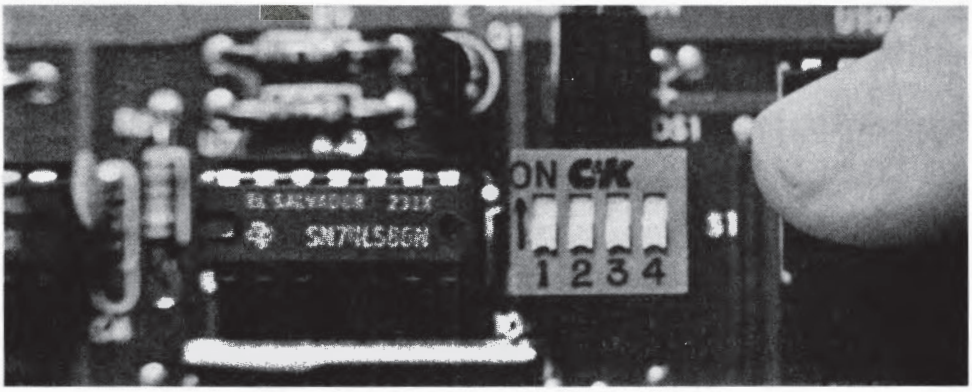


Fig. 14.1c DIP (dual-in-line package) switches provide direct mechanical control, but due to messy switching characteristics, usually should not be changed while the machine is on.

Optical Switches

Many solid state switching systems use "optical switches" which actually turn out to be close relatives of the BJT. Optical switches differ primarily in that they provide nearly perfect isolation between the controlling input line and the switched circuit. An optical switch is made up of a "photodiode/light-emitting diode pair." When current flows through a semiconductor, it glows (emits light; see Figure 14.1a). Similarly, when light of the proper frequency is shined onto a semiconductor, its resistance is changed.

The controlling input in an optical switch is a Light Emitting Diode (LED) and the switched circuit passes through a region of semiconductor (called a photodiode) which is exposed to the LED. The photodiode/LED pair are usually packaged together into a sealed plastic package that looks like an ordinary DIP (see Figure 14.1b). Whenever current flows through the controlling circuit, the LED glows, its light shines across a gap, and the photodiode in the switched circuit is turned on.

The substance of the photodiode is quite similar to the substance of a standard BJT, so all of the same limitations of unidirectional current, breakdown voltages, and On-Resistance still apply. All that is different is the isolation of the controlling input. No electricity passes between the two circuits, only light.

MOSFETs

There is a second kind of transistor called a Field Effect Transistor (FET; see Chapter 13) which is much better suited to a variety of switching tasks. In particular, the Metal Oxide Semiconductor Field Effect Transistors (MOSFETs) do quite well.

The most fundamental difference between a MOSFET and a BJT is that the input line which opens and closes a MOSFET is not in direct electrical contact with the switched circuit (see Chapter 13, Figure 13.3). It creates an electric field to alter the conductance of the semiconductor rather than pouring electrons or "holes" directly into it. This provides for much better isolation of the control input from the switched circuit although the isolation is not at all comparable to the isolation of an optical switch.

However, an additional consequence of the detailed function of MOSFETs is that they do permit bidirectional current flow. This means that they can be used to switch circuits that carry alternating current. You still have to keep voltages below about 20 volts to avoid destroying a standard MOSFET switch, and they do not tolerate very large currents, but they can be designed with very high off resistances. Further, just as with BJTs, there are special high power versions available with breakdown voltages in the range of 40 volts and current ratings in the range of one amp.

Mechanical Switches

For switching 120 Volt AC power, and for a variety of other high performance switched circuits, it's still hard to improve upon mechanical switches in which a piece of metal swings through the air to make or break a connection. Mechanical switches often have On-Resistances of a few tenths of an ohm, and nearly infinite Off-Resistances. In addition, the control input can be completely isolated from the switched circuit.

Mechanical switches are controlled either by direct action of the human hand (see Figure 14.1c) or by the pull of an activated electromagnet. Manual switches provide the optimum in full operator control, but, from a computer's point of view, switching speeds are slow and

erratic. An "electromagnetic relay" (see photo at front of this chapter) is also quite slow compared to a solid state switch, but the speed of switching is predictable and can be controlled directly by the computer. In fact, it is not uncommon to get operator control and predictable switching speed by using a manual switch to operate a relay.

When the control input to a relay is turned on, there is first a fixed amount of time before a sufficient magnetic field develops to cause the switch contact to begin to move. This "operate time" is followed by a "transfer time" during which the contact is swinging through the air. However, the worst part of the relay's performance often starts after the contact has hit home at its destination, because at that time it begins to literally "bounce," making and breaking contact hundreds of times before it finally settles.

Debouncing and Reed Relays

It is possible to design a relay with an operate plus transfer time in the range of one to two milliseconds, but settling times can continue for up to 50 milliseconds, thus greatly increasing total switching time. Therefore, methods for "debouncing" mechanical switches are crucial to their performance.

One example of a popular approach to debouncing is to use the output of the switched circuit to operate a special flip-flop which flips on the arrival of the current from the first contact and then ignores all subsequent on and off signals until it is reset. There are, in fact, a variety of debouncing chips (74279, 4043, 4490), but all of these have the limitation of requiring the switched circuit to pass through a semiconductor, so they may conflict with some of the best features of a mechanical switch (i.e., AC currents, high voltages, etc.).

A different approach to coaxing the highest possible performance out of a relay is to use lightweight contacts. These "reed relays" have good operate and transfer times because of the reduced mass of the contacts. If these "reeds" are wetted with mercury, then bounce can be sharply reduced or eliminated. Such "mercury wetted reed relays" have switching times in the range of one millisecond, require relatively low currents to activate their electromagnets, yet provide all of the advantages of mechanical switches.

Hall Effect Switches

A Hall effect switch is a sort of cross between a mechanical and a solid state switch. When a magnet moves, it can cause current flow in a nearby conductor or semiconductor. In one popular incarnation, all of the keys in a keyboard are constructed as magnet-tipped plungers. When a key is pushed, the motion of the magnet creates an electromagnetic field which alters the resistance of a nearby semiconductor. This has the effect of momentarily switching a solid state circuit element into its "on" state. Thus the control element is mechanical, but there is no swinging metal contact and hence no "bounce."

Counters, Timers and Clocks

In general, clock and counter devices are expected to be able to send out pulses at a variety of selected frequencies and to store up accumulating counts of these pulses. The keystone of the system is a small flake of quartz vibrating at a very high frequency. The counting device "divides" the original frequency by exact amounts so that slower versions are available at various points on the card. Most complete systems have both "decade" counters which divide the frequency by 10, and "binary" counters which divide it by two.

An example of such a system in its most generalized form is The Clock from Mountain Computer. The Mountain Clock has a one megahertz (one million pulses per second = 1 MHz) crystal, six decimal counters, and two 12 bit binary counters and a final one bit counter. The initial signal from the crystal pulses one million times a second, so the outputs from the six decade counters are 100 kHz, 10 kHz, 1 kHz (milliseconds), 100 Hz, 10 Hz and 1 Hz (one pulse per second), respectively. The first 12 bit counter sends a pulse every two to the eleventh seconds (5.69 hours), the second 12 bit counter pulses every 97 days (two to the 23rd seconds) and the last bit counter gives the 194 day pulse necessary to organize a full year of date outputs.

These counters serve a dual function. On the one hand, they act as frequency dividers for sending pulses, but at the same time they serve to accumulate counts of incoming pulses. All of the bits in The Clock's counters can be read from or written to by the 6502. When the second 12 bit counter receives its first pulse, it stores the number 1 to signify the passage of 5.69 hours. Three days later you'd expect to find 0000 0000 1101. This is equivalent to a decimal count of 13 and records the passage of 13 × 5.69 = 73.97 hours. You can set the clock before starting it by storing an initial number in the counter. For instance, 0000 1100 1110 could signify the passage of 206 × 5.69 = 1,172 hours in a year and hence represent the evening of February 17.

Most clocks and counters fall into two categories based on the degree to which they are either devoted to a single function, such as telling the time of day, or intended to provide a wide variety of clocking and counting functions. Most systems in both categories differ from the Mountain Clock in that they use specialized chips to handle most of the dividing and counting functions rather than building up from a large number of simpler separate chips.

Specialized Timing Chips

For simple time of day operation, it is possible to replace most of the circuitry on the Mountain Clock with a single small "clock chip" such as the NEC 1990c (Thunderclock) or 5832 (Applied Engineering's Timemaster and CCS' 7424 Calendar/Clock Module) and a much slower crystal. The clock chip has internal counters which permit it to send year, month, day of the week, date, hour, minute and second outputs. This approach makes for a simple, inexpensive card which does just one thing.

At the other extreme are flexible counter/timer systems based on the 6522, 6840, or AM 9513. These chips offer programmed selection of a wide variety of frequency division factors, multiple counters which can operate separately as well as in cascade, and which can accept incoming pulses from more than one unsynchronized external source.

The 6522 is called the Versatile Interface Adapter (VIA) and is used in several Apple based laboratory data acquisition systems (EcoTech, IMI, Interactive Structures) and is also available by itself on an interface card from Snave Systems called the Fly Board. In addition to counting, the 6522 has two bidirectional eight bit parallel ports with handshaking and a parallel to serial converter.

The timing functions are based on two 16 bit programmable counters and several latches to feed the counters. To program a four kHz output pulse you would store the number 250 in one of the latches and set the 6522 timing section into the appropriate mode. The 6522 could then use the one megahertz signal from the Apple bus to count out 250 microseconds, it would then send a pulse, reload 250 into the counter, and start counting down again. This sequence would repeat four times every millisecond, thus producing a four kHz output signal.

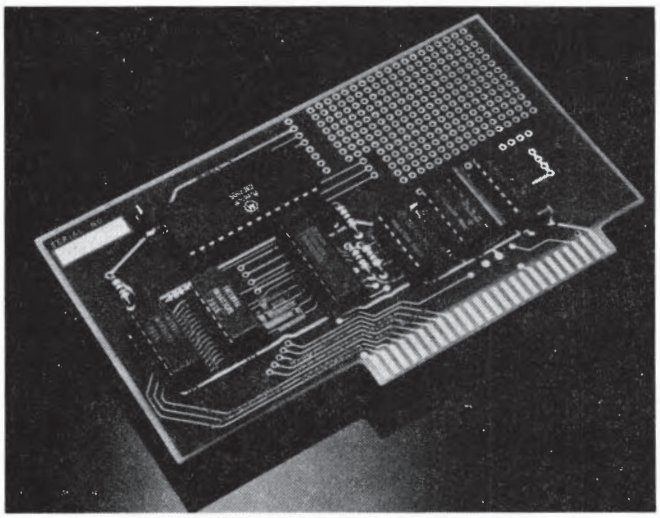


Fig. 14.2 CCS programmed timer uses a 6840.

There is a simpler device from Motorola called the 6840 which does not provide the additional digital I/O signals you get with the 6522, but which has three 16 bit counters. The Model 7440 Programmable Timer from CCS (see Figure 14.2) is based on the 6840.

The 9513 from Advanced Micro Devices (System Timing Controller)is devoted completely to timing functions. It has five 16 bit counters which can be operated separately or joined in groups. If all five were used as an 80 bit register to count a one megahertz pulse, the final bit would advance only once every 30 billion years. More reasonably, three of them set up as a 48 bit counter can serve as an eight year real time clock, and the remaining two can be used like the 6522 time registers.

The variety of control functions available on the 9513 permit the various registers and counters to be used to shape square wave patterns of some complexity, to insert various delays in output pulses, as well as to detect incoming signals for event counting and interval measurement. While these functions are very handy for several laboratory and industrial control chores, the versatility also makes the 9513 capable of handling most of the timing chores in a complete computer system. The designers of the Saybrook 68000 coprocessor board (see Chapter 30) chose to use the 9513 in this way.

Real Time Clock Cards for the Apple II, II+ and //e

The three important issues to keep in mind when you're choosing a real time clock card are your needs for precision and accuracy, ease of use from Applesoft and assembly language programs, and compatibility with ProDOS and time based commercial applications software. In addition, some clock systems come packaged with additional features which may sway your choice.

The Clock from Mountain Computer counts and announces time in divisions down to one millisecond and uses a good quality one MHz crystal system for .001 percent accuracy. This makes the Mountain Clock the only time card which is useful in demanding laboratory and industrial applications. All of the other Apple clock cards are oriented towards standard "time of day" tasks and so count time in divisions down only to one second.

There is a simple standard Applesoft programming approach for reading the time and date on Apple clock cards. Just issue an IN# statement for the slot the clock is in and then do an INPUT statement. The date and time pop right into your input variable just as if someone typed them in from the keyboard. The Mountain Clock documentation provides a variety of Applesoft, assembly language, and Pascal subroutines for doing fancy tricks with their clock, and these and other programs are also provided on a disk. The Thunderclock from Thunderware Inc. has less programming and hardware information spelled out in the documentation, but has a nice set of useful programs on disk, including Pascal routines.

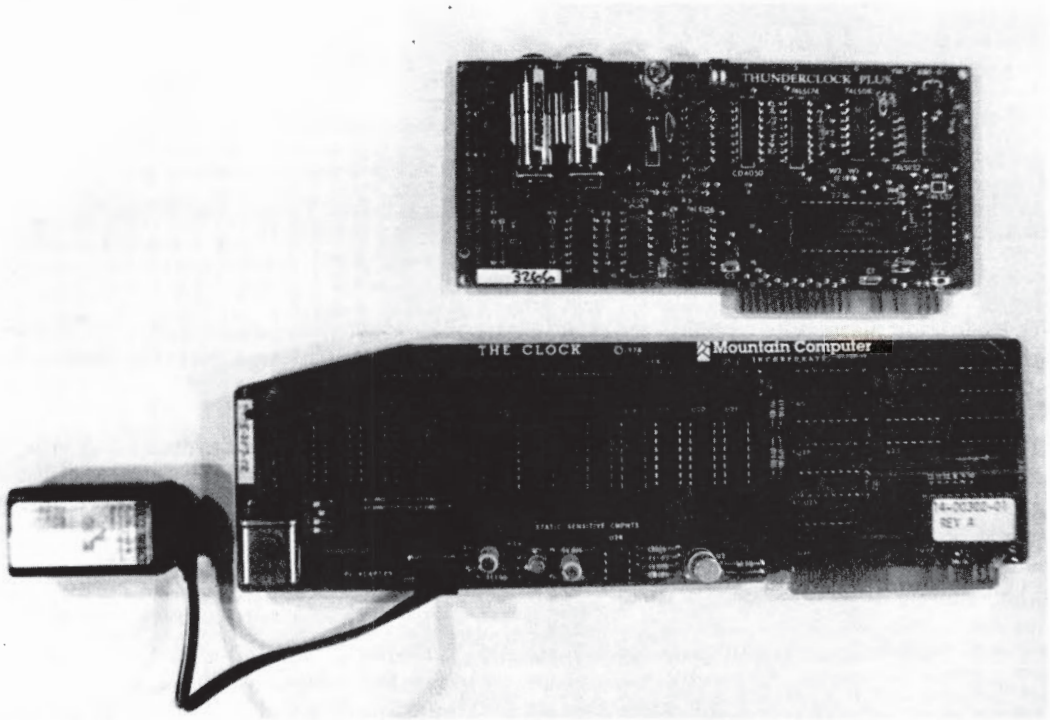


Fig. 14.3 Thunderclock (top) and Mountain Clock (bottom).

Time Format Compatibility

Aside from all other considerations, the compatibility problem may be the deciding factor for most folks. All of the clocks deliver a description of the date and time into the Apple's input line buffer at $200 (see Chapters 21 and 33), but the exact form of the description varies. The standard format is the Mountain Clock time string which is: 05/12 08;32;27.345 (for a little after 8:30 a.m. on May 12). Other clocks such as the Thunderclock, the CCS Calendar/Clock Module, and the Applied Engineering Timemaster offer a "Mountain Clock mode" in which the card sends a string which is formatted identically to the Mountain time string except that the milliseconds are set to 0.

However, the Thunderclock, CCS module and the Timemaster are also capable of providing the day of the week and a.m./p.m. status. Further, the Timemaster and CCS module also announce the year. The Thunderclock has one more mode which isn't easily available on the other cards, which returns the data in numeric format: 05,03,12,08,32,27 (for Tuesday, May 12 at 8:32:27 a.m.). This is convenient in some programs because the values can be read directly into numeric variables without the use of strings.

ProDOS Compatibiiity

The most important compatibility issue has to do with ProDOS, which works automatically with the numeric format of the Thunderclock but not with most other clock cards. According to the author of ProDOS, this choice had mostly to do with the fact that he just happened to own a Thunderclock card. The upshot of this is that even if your clock has a "Thunderclock mode," ProDOS probably won't be able to read it. The ProClock from Practical Peripherals is a newer product that is completely compatible with the ProDOS clock routines. This is not an insurmountable problem since any clock manufacturer should be able to provide you with a "patch" to ProDOS to let your own version of ProDOS read your own clock.

(For hackers who would like to know: a time request in ProDOS jumps through a vector stored at $BF06 on the global page. If a Thunderclock is installed, ProDOS will detect this fact from the signature bytes in the Thunderclock ROM, and it will set the time vector to point to a routine, in bank switched RAM, at $F142. At $F149, this routine puts a $A3 (ASCII #) in the accumulator and does a JSR to $Cn0B to set the Thunderclock to numeric mode, followed by a JSR to $Cn08 which causes the time to be read into the input line buffer at $200. The remainder of the routine manipulates the values into the compressed format specified in the ProDOS manual (see Chapter 37) and stores it in the global page at locations $BF90 to $BF93.)

Interrupt Generation by Clock Cards

Several clock cards can automatically generate interrupts (see Chapter 27) at a selected frequency. One common use of this feature is to seize control of the Apple once a second and update a time display on the screen. The Mountain Clock will accept a command to send interrupts once a second. If you want interrupts at a different frequency, you have to do a "cut trace and solder" modification on the board. Mountain gives instructions for getting interrupts at 37 different frequencies ranging from 10 kHz (every 0.1 milliseconds) to once a month.

The CCS Module and the Timemaster can generate interrupts at one kHz (actually 1,024/ second), once a second, once a minute or once an hour. You use jumpers to hardwire the interrupt configuration on the CCS card, but the Timemaster has a Peripheral Interface Adapter (PIA) which lets you set up the interrupts from software. The Thunderclock has three interrupt frequencies which can be set from software: 64 Hz, 256 Hz, and 2048 Hz.

Although the high frequency interrupts from the Mountain Clock, ProClock, CCS Module, and Timemaster may be useful in some demanding applications, none of these cards provides the most important interrupt frequency for most applications. The interface card for the Apple II Mouse generates a 60 Hz interrupt tied to the vertical blanking signal (VBL) of the Apple video generator (see Chapter 9). This 60 Hz VBL interrupt is important because it lets a programmer cut into a program to rearrange the video display while the video electron beam is turned off. This is important for good quality animation as well as for "windowing" in multitasking operating systems.

In the //c, the IOU chip on the motherboard can be configured to generate VBL interrupts at 60 Hz (see Chapter 9). This interrupt can't be used as part of a software clock, however, because the IRQ line to the 65C02 must be disabled whenever the disk drive is reading or writing data.

Additional Features on Clock Cards

The Thunderclock card has an interface system for operating a BSR X-10 controller. This means that you can program light switches, coffee makers, etc., around your home to turn on or off at selected times. Multifunction cards from Artra and Prometheus also offer BSR X-10 controller interfaces.

A BSR system connects to the 120 VAC power available at the wall sockets throughout your house and can transmit the electrical equivalent of ultrasonic tones over the AC power lines. Plug-in modules at sockets throughout the building can detect the tones. Each plug-in module is assigned a tone sequence which it can recognize and respond to. Thus providing the Apple with a means of giving instruction to a BSR controller extends its control capabilities throughout the building without the need for installing any new cables.

The Artra Waldo card has voice recognition, voice synthesis (see Chapter 11), and an audio amplifier packed on board along with the clock/calendar and BSR controller. In theory, this means that you can tell it verbally when it should turn on the lights, etc., and it can also be instructed to yell out messages at selected times. Voice recognition and voice synthesis are not completely "mature" technologies, so you shouldn't get your hopes up too high for this device, but it can do remarkable things.

The Prometheus Versacard is much less glamorous, and less expensive. It includes a graphics parallel printer port, a serial port, and a BSR interface along with a clock/calendar. The Mountain Computer CPS Multifunction Card has a parallel and serial port as well as a one second resolution clock, but no BSR interface. The Versacard and the CPS card both use the "phantom slot" technique (see Chapter 22) to load several different input/output capabilities into a single slot while tricking the Apple into thinking that the various functions are spread around in several different slots.

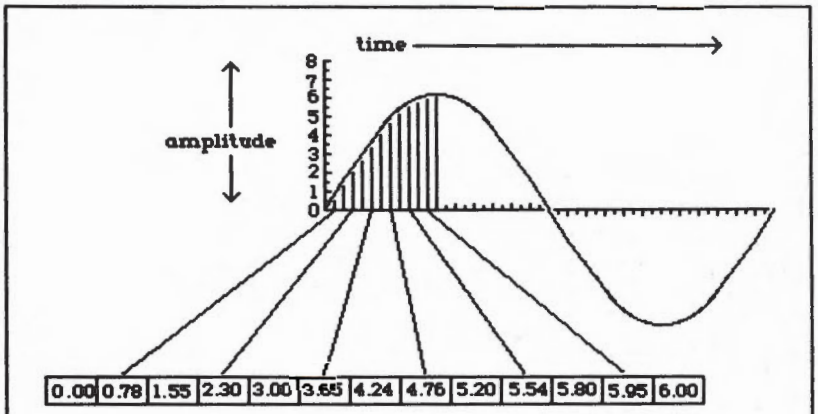
External Clocks for the //c

Both Hayes and Prometheus make external clocks which can be interfaced with the Apple via a serial port. External clocks don't make much sense for an Apple II, II+ or //e, but they are your only option for a //c. The Hayes Chronograph is a fairly expensive external device which has the sole advantage for Apple owners that it, like the Pro Modem 1200, doubles as a digital clock with a display you can glance at whenever you want to know the time.

The Prometheus Pro-Modem 1200 is a bit more interesting. This device includes a clock/calendar inside a 1200 bps modem (see Chapter 17). It can be modified to generate interrupts via the serial port (see Chapter 16) and you can get a ProDOS patch which reads it automatically. Since the ProModem is a comparatively inexpensive modem in the first place, you can think of it as providing a clock for free and thus it makes very good sense as a //c peripheral. You should keep in mind, however, that most of the commercial software which can use real time information (i.e., Visidex, DB Master, Executive Secretary, Transend, etc.) can only operate internal clock cards and that ProDOS expects to find the Thunderclock, so an external clock may only be useful if your software can be properly configured.

Analog to Digital Conversion

The great bulk of electronic monitoring and measuring systems used in audio, laboratory, and industrial tasks have been designed to represent phenomena in the physical world as patterned analog waveforms. These waveforms are usually expressed as voltages which vary through time. Analog to Digital Conversion is a process which links the analytical power of digital computers to the sensitivity and versatility of analog instrumentation equipment.



During "DIGITIZATION," the amplitude of a waveform is sampled at regular intervals. The resulting numbers can be stored in memory as a digital representation of the original waveform.
This is the task performed by an "Analog to Digital Converter."

A "16 bit" ADC would divide this range into 65,535 steps, from $0000 to $FFFF.
Each 16 bit sample is represented in memory by a two byte number as shown below:

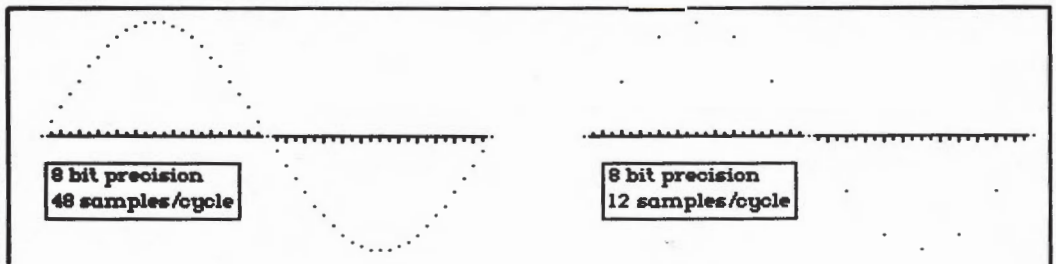
$8000	$8C76	$98CD	$A4CD	$B000	$BA66	$C3D7	$CC29	$D333	$D8A4	$DCCC	$DF33	$E000

An "8 bit" ADC could measure only 256 different amplitude levels. This precision is more than adequate for the waveform shown above, and it takes up less memory:

$80	$8C	$98	$A4	$B0	$BA	$C3	$CC	$D3	$D8	$DC	$DF	$E0

Reducing the "sampling rate" from 48 samples per cycle to just 12 samples per cycle saves even more memory, but still produces a useful description:

$80	$B0	$D3	$E0



Later, the numbers can be recalled from memory and used to regenerate a close model of the original analog waveform.
This is the task of a "Digital to Analog Converter."

Fig. 14.4 A/D and D/A conversion.

The most convenient mental image of an analog waveform is based on a curve drawn on graph paper. The steps along the X-axis represent increments of time. The curve can be described as a series of Y values, with one Y value assigned to each increment along the X-axis (see Figure 14.4). The Y values are "samples" and the distance between the samples reflects the "sampling rate." If the sampling rate is too slow, then the Y values won't provide a very good description of the curve.

The precision of the sample reflects the smallest variation which can be detected. A great deal of information can be lost if the precision of the sampling is not adequate to the complexity of the waveforem. For most Analog to Digital Converters (ADCs), there is a conflict between precision and sampling rate. The more precise the Y measurement, the more time the ADC requires to finish making a sample, and hence the slower the sampling rate. The objective in the design of an ADC is to get the greatest precision with the fastest sampling rate.

There are three major kinds of devices used to carry out analog to digital conversion. Integrating ADCs are simple, inexpensive and relatively low in performance. "Digital Servo" ADCs can achieve fairly high performance at a modest price. Finally, there are "Flash" or "Parallel" ADCs which do effectively instantaneous conversions but in which the conflict is between precision and high cost; a high precision flash ADC can cost over $1000.

Integrating ADCs

An integrating ADC does not measure voltage directly. Rather, it relies on currents generated by the voltages of the waveform. A current is a rate of flow of units of charge. Conversion is based on counting units of charge, or detecting the rate at which they are flowing. These devices can be built out of fairly simple components such as capacitors, resistors and "comparators." (A comparator is a kind of "op amp" (see Chapter 13) which can detect equivalent signals on its two inputs, and it is used as a component in all of the different kinds of ADCs.)

The first step in conversion is to use the voltage in the waveform to create a current. This requires that the voltage input be connected to a resistor. The current flows into a capacitor which accumulates units of charge. In "sample and hold" types of converters, the waveform input is connected to the capacitor for a brief, fixed amount of time. Higher voltages cause greater currents and hence can load more units of charge into the capacitor during the sampling interval.

In a "dual slope" integrating ADC, the charged capacitor is then disconnected from the voltage source and allowed to discharge to ground at a fixed rate of current flow. A timer is started when the discharge begins, and a "comparator" detects the moment at which the capacitor is fully discharged. The amount of time for the discharge is proportional to the input voltage. The time count is therefore a numerical representation of the average voltage during that sampling interval.

The Apple II, II+ and //e have four ADCs connected to the game port (just two in the //c) which are similar in concept to a dual slope ADC. In the Apple game controller system, a fixed five volt signal is used as the input signal, but the resistance, and hence the current, is varied. Turning a game paddle knob in one direction gives a high resistance and slow rate of current flow, while the other direction gives a low resistance and high rate of current flow. The incoming current charges a capacitor until the charge becomes equal to a reference charge stored in a second capacitor. The amount of time for the input capacitor to become fully charged is proportional to the setting of the resistor in the game paddle (see Figure 14.5).

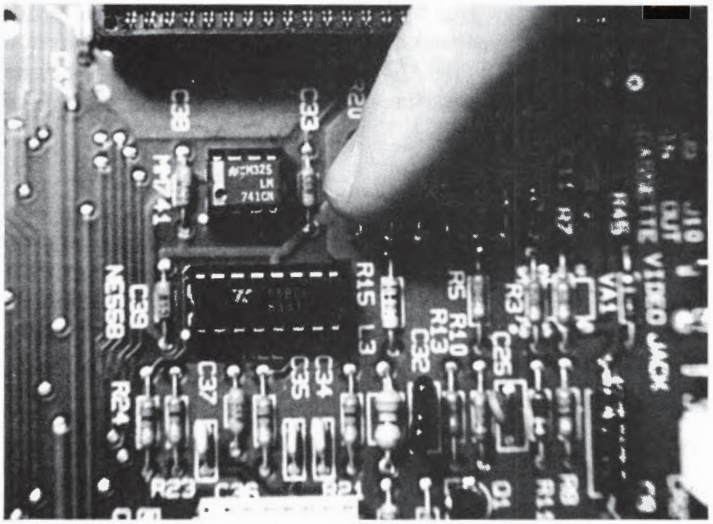


Fig. 14.5 Built-in A/D converter on motherboard for game paddles. The system involves the NE558 timer; resistors R21, R22, R23 and R24 with capacitors C34, C35, C37, and C40 for the four different paddles, and C36 for setting the threshold voltage.

There is a subroutine in the Monitor (see Chapter 21) which begins the sampling process and then checks the status of the ADC every 10.8 microseconds. If the external resistor is set near zero, the ADC is triggered very rapidly (the actual minimum for software detection is about 15 microseconds). When it is at the opposite maximum, it can take up to 256 test periods or 256 times 10.8 equals 2.76 milliseconds to determine the setting. This process is explained in more detail in *The Apple Circuit Description* by Gayler (Sams Publishing) and in *Understanding the Apple II* by Sather (Quality Software).

This kind of ADC is inherently slow because the sampling rate must allow for the worst case situation in which there is a full sweep from zero to the maximum charge. An alternative is to detect the change from the previous sample rather than the difference from zero. In many waveforms, the change from sample to sample is always quite small compared to the total difference between minimum and maximum amplitude. The idea is to "follow" the input voltage and make only incremental changes in the test signal examined by the comparator.

There is a kind of integrating ADC called a Voltage to Frequency Converter (VFC) which is based on an attempt to follow the incoming signal and continuously adjust the amount of charge being dumped into a test capacitor. The system is based on a device which sends pulses of charge at a controlled frequency. When the frequency is increased, more charge is dumped into the test capacitor per unit time, thus simulating an increase in current. The comparator senses the difference in charge between the input capacitor and the test capacitor and uses this information to either increase or decrease the frequency setting of the charge generator. The frequency setting is available at all times during this process as a digital number that can be read by the computer.

The principal limits on sampling rate are the speed at which the comparator can detect changes and the maximum rate at which the charge generator can alter the frequency of its output. The speed at which small changes are detected and balanced usually determines the sampling rate. There are comparatively inexpensive VFCs which can manage sampling rates at speeds up to one megahertz.

However, depending on the shape of the waveform being measured, the maximum rate at which the charge generator can respond to large changes in the input may be the more

important determinant of performance. The "slew rate" of the system is the maximum rate of change in volts per second that the converter can track accurately. The best VFCs have slew rates in the range of tens of thousands of volts per second. More practically, this means that they can respond to a change of one volt in a few thousandths of a second.

Digital Servo ADCs

The key component in a digital servo ADC is a Digital to Analog Converter (DAC; see below). The DAC is the functional opposite of an ADC; it responds to a digital number by producing the desired voltage. The underlying designs of the various kinds of digital servo ADCs are very similar to the designs of integrating ADCs except that there is no need to deal in currents and charges and balanced capacitors. A digital servo ADC can actively produce test voltages which can be compared directly to the voltages of the waveform.

There are "staircase," "tracking," and "successive approximation" digital servo ADCs. The staircase and the tracking ADC are similar in concept to the dual slope and the VFC respectively, while the successive approximation ADC takes unique advantage of digital servo technology.

Staircase Converters

In a "staircase" type of digital servo ADC, the converter requires a Sample and Hold Amplifier (S/H Amp) to acquire a sample input voltage and then hold it steady during the conversion process. The staircase ADC then begins loading numbers into its control register, starting at zero and stepping upwards one by one. The DAC part of the converter examines the contents of the control register and produces the requested test voltage. If the test voltage is not equal to the input voltage being held by the S/H Amp, then the number in the control register is incremented by one, the DAC produces a higher test voltage, and so on.

This process shares the same limitations which hinder the performance of dual slope integrating ADCs. The staircase must start at zero and step upwards each time. The staircase device operates much more quickly than the dual slope device, but it does not take full advantage of the potential speed of the digital servo systems.

Tracking Converters

A "tracking" ADC is able to follow small changes in the waveform. The key to this feature is that it can decrement the number in its control register as well as increment it. This relieves it of the need to sweep up the "staircase" from zero for each measurement. A good tracking ADC can operate without an S/H Amp. The control register is continually updated and the value it contains is available to the computer as frequently as the computer can read it.

The sampling rate can be pushed up into the range of three to five megahertz. As with VFCs, slew rate may be the major determinant of the kinds of waveforms it can describe accurately. Because this is a digital device, however, it is possible to improve the slew rate by adding larger numbers to the control register rather than just incrementing by one. Such a modified tracking ADC can achieve slew rates in the range of five million volts per second (5 volts per microsecond).

Successive Approximation

The most popular type of digital servo ADC is the Successive Approximation (SAR) converter. Like the staircase converter, a SAR requires a good S/H Amp and starts from scratch for each sample. However, it uses a much more efficient algorithm for setting its control register.

A staircase converter starts with its Least Significant Bit (LSB) and adds one bit at a time. For an eight bit converter, this might mean that it tries 0 millivolts, and then one millivolt and then two millivolts, etc., until it had made the full 256 tests to completely cover its range. Conversion time varies according to the magnitude of the input voltage, and the worst case conversion time is 256 times the testing interval.

A SAR converter begins by testing with its Most Significant Bit (MSB). For an eight bit converter this might mean that it first tries 128 millivolts. If this is too high, it sets the MSB to 0 and tries the next highest bit which would call for 64 millivolts. Each step divides the range of possibilities by half, and the process is always finished after the eighth test. The result of this algorithm is that all conversions take the same amount of time, which is, for example, eight times the testing interval for an eight bit converter.

The SAR algorithm becomes extremely important for high precision conversion. If the testing interval were one microsecond, a 16 bit staircase converter could take up to 65,536 microseconds to do the conversion, but a similarly built SAR converter would always be done in 16 microseconds.

Flash ADCs

A flash ADC provides nearly instantaneous conversion because it cuts out all of the adjustment steps required by other methods. Instead of using a single comparator and stepping the test voltage through 256 levels, an eight bit flash ADC has 256 comparators, each permanently set to a different voltage level, all of which are exposed to the incoming signal at the same time.

If the comparators are preset to a series of voltages at one millivolt intervals (i.e. 0 millivolts, 1 millivolt, 2 millivolts, etc.) then an incoming voltage of 184 millivolts would simultaneously activate the first 184 comparators. The array of comparator outputs that are turned on are then channeled through a set of logic gates which can express the number of active comparators as an eight bit digital number. The actual conversion time for a flash ADC is limited by the speed at which the semiconductors can respond, so that 20 nanoseconds is a practical conversion time.

The number of comparator amplifiers required in a flash ADC is determined by how many bits of precision are desired. There must be one comparator for each possible value in the range, so that a 12 bit flash ADC requires 4,096 comparators. A 12 bit flash ADC is therefore a very difficult part to manufacture, and so the cost goes up into the thousands of dollars.

Digital to Analog Conversion

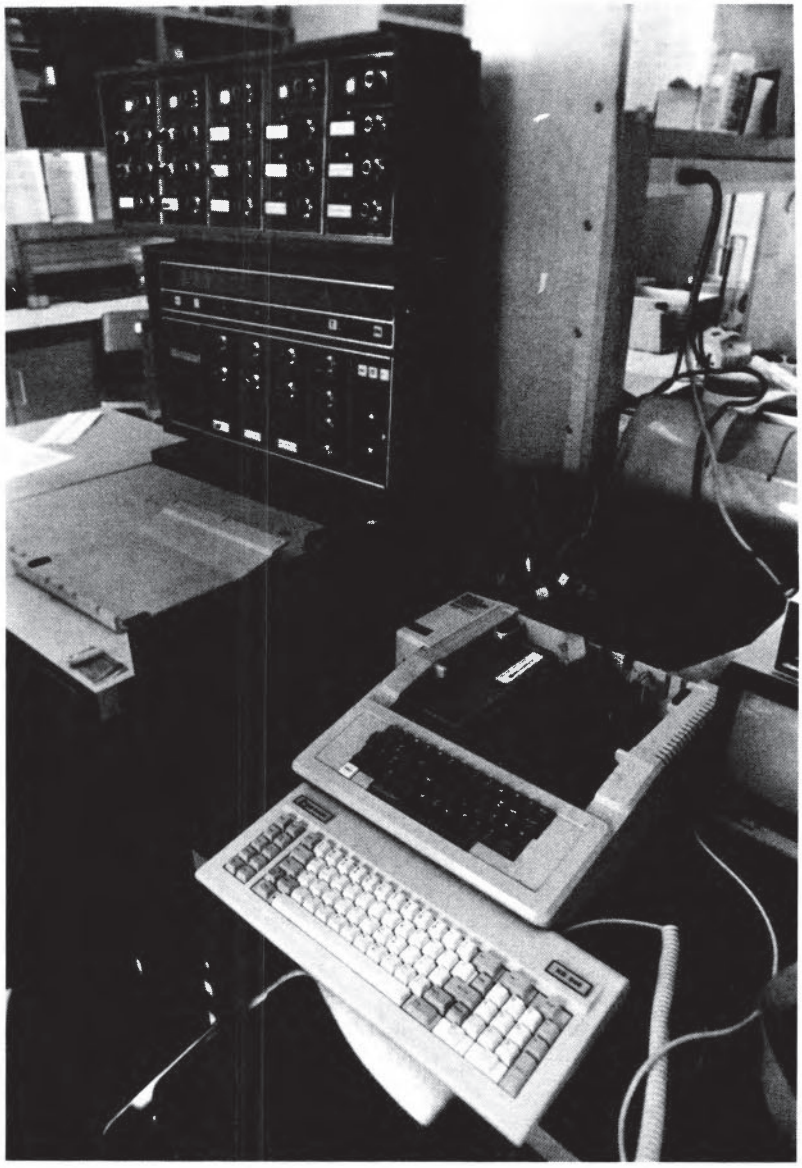
To gain full participation in the analog world which surrounds it, an Apple must be able to generate analog waveforms of its own. Some tasks such as reproducing the human voice or playing out music require high speed variation, good precision, and close modeling of the original waveform over a wide range, i.e., good "linearity." Other tasks such as managing the fine adjustment of motor speeds and the setting of variable controls on analog devices make demands on accuracy (i.e., the achievement of an exactly correct voltage compared to some external reference) and on the range of output voltages and currents the Apple can make available.

Most devices which carry out digital to analog conversion are similar in concept to a flash ADC, but work in the opposite direction. The usual arrangement is to provide an array of current sources, each set to a different level, and then to use a digital input to select which of the sources will be connected to the output.

A DAC also differs from a flash ADC in that an eight bit DAC requires just eight current sources while an eight bit flash ADC requires 256 (two to the eighth) comparators. The reason for this difference is that the circuits of the DAC can sum the currents after they are selected and before they are output, while a flash ADC has no way of subtracting or dividing prior to conversion. In fact, it is because of the need to sum the outputs in a DAC that current sources are used rather than voltage sources.

In a good current switching DAC, the actual selection and current summing is extremely fast—typically on the order of 350 nanoseconds. A great deal of the speed is lost subsequently in the conversion of the summed current to a voltage by a current summing operational amplifier (op amp). Typical conversion speeds run in the range of three to five microseconds for voltage output.

To vary the output range of the DAC, there are two approaches. In a "multiplying DAC," a range selector can alter the voltage which drives the array of current sources. The DAC and its output are unchanged, but the magnitudes of the currents flowing in the system are varied. The alternative, and more popular approach, is to put a gain amplifier in line to amplify the output of the current summing op amp.



Chapter 15

Laboratory Data and Control Systems

There is virtually no limit to the scale, scope and resolution of microcomputer based systems for collecting data, monitoring processes and carrying out complex control tasks. If you're in a position to having fairly good confidence about the electronics involved and know exactly what you need, you can often do quite well at a low cost with a single board product. However, the most elaborate systems are available as turnkey setups complete with data analysis software, local service representatives and experienced technical consultants.

Several companies sell board level products which specialize in analog to digital conversion (Hollywood Hardware, Applied Engineering, RC Electronics, Northwest Instruments, Interactive Structures, Data Translation), and others manufacturers include ADCs with other functions on a single board (Mountain Computer, MicroDimensions, IMI). These systems offer the full range of ADC performance at much lower cost than a full scale data acquisition system.

The systems range in performance from dual slope converters with a 50 millisecond conversion time (IMI's Adalab) to flash converters which need just a little more than 50 nanoseconds (RC Electronics' HS 7 Applescope; see Table 1). Most manufacturers offer 8 or 12 bit precision, but there are also 14 bit (RC Electronics' HR 14 Applescope, Data Translation, Keithley/DAS' ADM2) and 16 bit (Data Translation, Cyborg's ISAAC I-150) converters available.

Software Speed

The performance of any of these products has as much to do with the quality of the support software as with the hardware specs. In all of the systems except the RC Electronics Applescopes and the Cyborg ISAAC 2000, the maximum speed of data collection is determined by the program execution speed of the Apple rather than by the ADC hardware. As each data byte is collected, the 6502 must get the byte from the data port and place it in RAM memory.

Most of the packages allow data collection at up to 10 kHz (equivalent to 100 microsecond conversion time), while the fastest machine language routines (Keithley, RC Electronics) can push this up to somewhere near 30 kHz (equivalent to 33 microsecond conversion time). A hardware system based on DMA (see Chapter 27) could bypass the 6502 and push the speed up to one MHz, but only the RC Electronics ADC systems are fast enough to require this sort of loading speed, and those folks use a simpler solution for very high speed data collection up to 14 MHz.

Many of the manufacturers provide additional commands for Applesoft BASIC all built around the ampersand (&) command (see Chapter 38). This makes it easy to issue configuration commands and collect data from within an Applesoft program. The ampersand commands can call up high speed machine language routines to actually move the data before returning to Applesoft.

Interrupts for Efficient Use of the CPU

The most elegant software packages offer "background/foreground" operation based on interrupts (see Chapter 27). This means that the Apple can go along doing statistics and printing out plots yet still be able to temporarily snap its attention over to the input ports whenever a byte of data arrives. The AMPRIS software from Interactive Structures and the Soft 500 software from Keithley both have this ability.

In fact, this is no trivial feat, because Apple DOS 3.3 has a serious bug in its interrupt handling (see Chapter 27), and AMPRIS has to correct this bug by reconstructing several segments of DOS 3.3. This background/foreground feature may turn up more often in future software from other manufacturers since ProDOS is designed to handle interrupts properly without any Herculean effort on the part of the programmers. There is a new set of ROMs available for the //e which correct the interrupt bug, so many more programmers will be able to create interrupt driven software in the future.

Complete Software Acquisition and Control Systems

A further consequence of software considerations arises if you suspect you may eventually need more than just analog to digital conversion (i.e., timers, analog output, digital interfaces, etc.). You are much better off buying a system in which a single software environment supports several different kinds of interfaces. The Interactive Structures system is strong in this regard since you can begin with a single relatively inexpensive board, but retain the capability for substantial expansion later, all within a single software framework.

The more expensive full scale data acquisition systems from Keithley, Cyborg, Eco-Tech and Interactive Structures all use external cabinets for installing an assortment of interfaces including a variety of ADCs which the buyer can mix and match to suit particular needs. Further, they all offer fairly sophisticated data collection software packages.

High Performance ADC Cards

High performance, modestly priced ADC cards include the AI13 from Interactive Structures, the AD-121602 from Hollywood Hardware, and DT2832 and DT2834 from Data Translations. These are SAR converters (see Chapter 14) with 12 bit precision (and optional 14 bit or 16 bit precision for the Data Translation boards), and all offer software switching among 16 channels of input. Of the three, the Hollywood Hardware board (see Figure 15.1a) is the least expensive and offers a few features in addition to A/D conversion, but the Interactive Structures and Data Translation boards have slightly better performance on several features pertaining to A/D conversion itself.

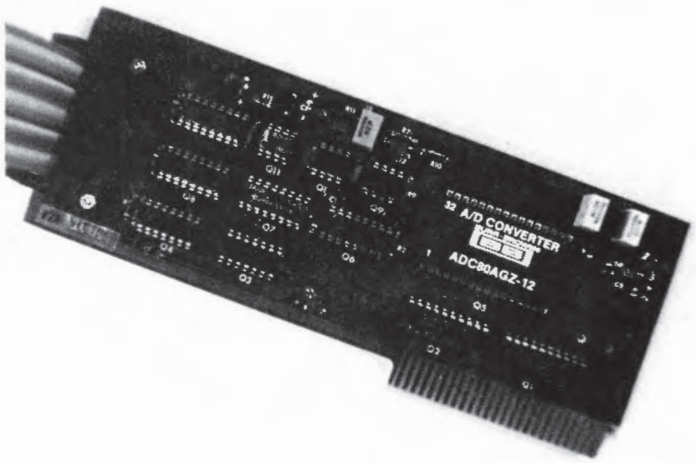


Fig. 15.1a Hollywood Hardware AD121602 analog to digital converter.

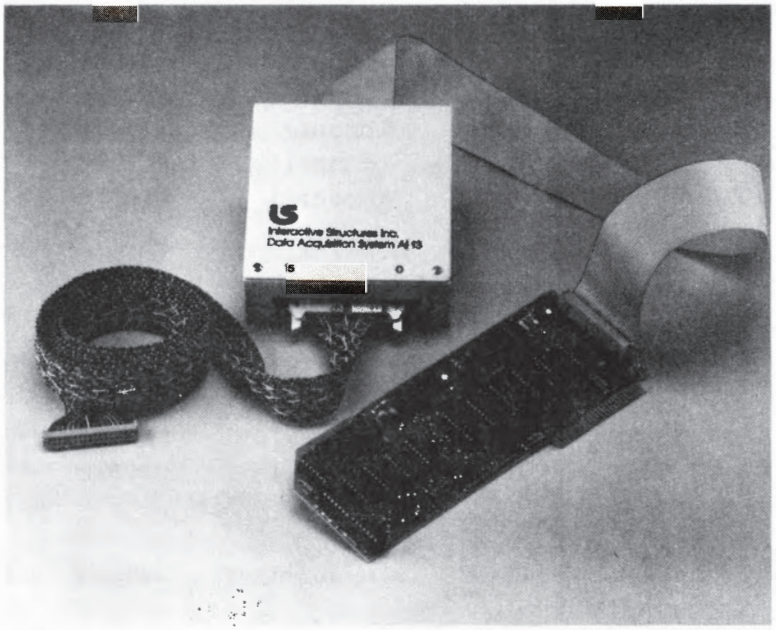


Fig. 15.1b DAISI AI/13 analog to digital converter from Interactive Stuctures.

The conversion time for the AI13 (see Figure 15.1b) is 20 microseconds, while the Hollywood Hardware board comes in a little behind at 25 microseconds. For both of the boards, sampling can proceed at full speed on a single channel, but there is a delay of about 100 microseconds whenever you switch to another channel.

The major differences are in the quality of the "input range" and "input impedance." It is necessary to choose a range of input voltages that the ADC can expect to see from an input waveform. You select the range and the ADC divides it into parts, 256 parts for an eight bit ADC, 4096 parts for a 12 bit ADC, etc. The AI13 can switch among four unipolar input ranges (0-5 Volts DC, 0-1 VDC, 0-0.50 VDC, and 0 to 0.1VDC) and four bipolar ranges (+/− 5 VDC, through +/− 100 mVDC), all under software control. The Hollywood Hardware board has just five input ranges (+/− 10 VDC, +/− 5 VDC, +/− 2.5 VDC, 0-10 VDC, and 0-5 VDC), and these must be set by jumpers on the card.

Analog to Digital Converter Specs.

Applied Engineering				
	8 bit	78 µs	8 channel	$129.00
Cyborg				
ISAAC I-130	12 bit	25 µs	1 channel	$600.00
ISAAC I-100	12 bit	25 µs	16 channel	$850.00
ISAAC I-150	16 bit	60 µs	2 channel	$1,100.00
ISAAC I-180	12 bit	35 µs	2 channel	$700.00
ISAAC 21A	16 bit	10 ms	2 channel	$1,700.00
ISAAC C-100	12 bit	5 µs	4 channel	$3,825.00
Data Translations				
DT 2832	12 bit	50 µs	16 channel	$700.00
DT 2832 (5716)	16 bit	200 µs	16 channel	$1,700.00
Eco Tech				
ALIS A/12	12 bit	20 µs	16 channel	$1,200.00
ALIS A/08	8 bit	70 µs	16 channel	$935.00
Hollywood Hardware				
AD 121602	12 bit	25 µs	16 channel	$450.00
Interactive Microware				
Adalab	12 bit	50 ms	1 channel	$495.00
Interactive Structures				
A/02	8 bit	70 µs	16 channel	$299.00
A/13	12 bit	25 µs	16 channel	$530.00

Keithley DAS				
ADM1	12 bit	25 µs	1 channel	$700.00
ADM2	14 bit	35 µs	1 channel	$950.00
Micro-Dimensions				
µD-1000	8 bit	120 µs	8 channel	$300.00
Mountain Computer				
A/D	8 bit	9 µs	16 channel	$350.00
Northwest Inst.				
Model 85	8 bit	40 µs	1 channel	$995.00
RC Electronics				
APL-D2	8 bit	2 µs	1 channel	$795.00
HR14	14 bit	14 µs	1 channel	$995.00
HS7	7 bit	500 ns	1 channel	

Table 15.1 Analog to digital converter specs: precision, conversion time, number of input channels, price.

The input impedance reflects the fact that the ADC looks something like a resistor from the point of view of the signal source. If the input impedance is low, then the ADC will draw a lot of current from the source of the waveform, and possible alter the appearance of the waveform as a result. A high input impedance minimizes the effect of the ADC on the circuit it is supposed to be sampling from. The input impedance for the AI13 is 10 MegOhms, while the Hollywood Hardware board is rated at 150 KOhms.

For many applications, the small margin of extra performance on the part of the AI13 may not be worth the additional $100. The extra features on the Hollywood Hardware board are a tracking reference signal, and also two single bit inputs. An advantage of the AI13, on the other hand, is that it is just one part of a family of nearly a dozen products in the DAISI series from Interactive Structures (see below), and therefore is the best choice for a system which may be expanded in the future. Although both systems offer ampersand commands, the AI13 software is a little more versatile.

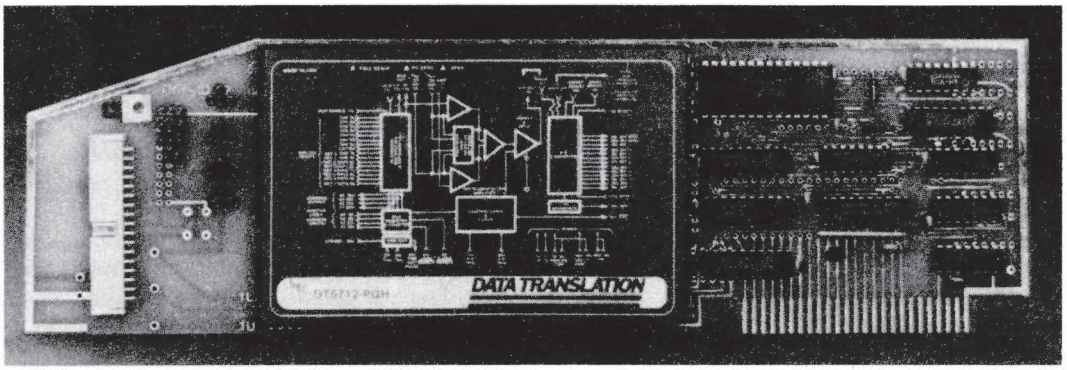


Fig. 15.2 Data Translation DT2832 analog to digital converter.

The Data Translation boards (see Figure 15.2) have a very high performance input amplifier, an external trigger, and, with the DT2832, a programmable timer. One advantage of the Data Translation boards is that you have the option of getting 12, 14, or 16 bit precision. With the DT2832, the data collection rates are 20 kHz, 10 kHz, and 2.5 kHz for the three different precision levels, while the same figures for the DT2834 are five kHz, 800 Hz, and 200 Hz.

Single Card Laboratory Interfaces

An A/D converter chip is sufficiently compact that it can share the space on an interface card with several other interface devices. The major problem is that there is limited space for all of the inputs and outputs if each of the devices has multiple channels. Accordingly, Interactive Microware Inc. (IMI) has designed their Adalab card with a one channel, 12 bit A/D converter, a one channel, 12 bit D/A converter, and a 6522 (see Chapter 14) to provide two parallel ports and two 16 bit timer/counters.

The Adalab A/D converter is a very slow "dual slope" integrating converter (see Chapter 19) with a 50 millisecond conversion time, but the D/A converter has a 20 microsecond conversion time and sources up to 10 milliAmps. The IMI data collection, display, printer dump, and plotter software is fairly extensive, and it is possible to operate the Interactive Structures AI13 with the IMI software as a system expansion. In addition, IMI offers an optional reed relay multiplexer which goes outside the Apple for slow switching among eight channels, an instrumentation amplifier for conditioning signals on the way to the A/D converter, and a buffering system to give added drive to the 6522 digital outputs so they can operate relays.

MicroDimensions makes a more modest lab interface called the MicroD-1000 which has an eight channel, eight bit ADC with a conversion time of 120 microseconds, and a 6821 peripheral interface adapter which provides eight bits of digital input and eight bits of digital output. The manual for the MicroDimensions system tells you all about how to build your own optically isolated input system and power relay controls, but you have to do all that yourself with parts from your local electronics supplier.

Options for Eight Bit ADC boards

If you need fast conversion but are willing to concede a little on precision, then the Mountain Computer A/D + D/A board could be a good choice. It has an eight bit ADC with a conversion time of just nine microseconds. This is a 16 channel converter with an input impedance of one MegOhm. You probably can't get your software ready for a new data point more than once every 20 or 30 microseconds, but the fast conversion time helps avoid averaging a signal. The digital to analog converter mounted on the same board is a 16 channel, eight bit system with a slew rate of 10 volts/millisecond and an output range of +/- 5 volts.

There is also a 16 channel, eight bit ADC from Interactive Structures (AI02), but it has a conversion time of 70 microseconds. The Mountain Computer board sells for $350 and includes both A/D and D/A conversion, while the AI02 is $300 for A/D conversion only. Applied Engineering makes an eight channel, eight bit ADC with a conversion time of 78 microseconds. It has a rather low input impedance (20K Ohms) and is not backed by strong software, but it does sell for a modest $129.

Digital Oscilloscope Cards

A standard analog oscilloscope traces a waveform on a CRT screen, using the voltage of the input signal to control the Y position of the beam, and some time base to control the X position. Analog "storage" oscilloscopes let you retain the image on the screen; however, to save a waveform for later analysis you usually attempt to capture the incoming signal on a high quality tape recorder or on the paper of a strip chart recorder at the same time that you are also creating an image of the signal on the screen. If you store on tape, you can do the analysis by passing the signal back out from the recorder, through an A/D converter such as the Interactive Structures AI13, and on into the Apple for digital storage on disk.

A top quality analog oscilloscope can follow signal components at up to 400 MHz, and a good 16 channel FM tape recorder can sock away hundreds of millions of points per second, all available for later playback and A/D conversion at slow speed. What you can't get from such a system is "real time" analysis. In the past few years, "digital storage oscilloscopes" have become a popular means of getting instant analysis of some kinds of signals. In these systems, the original incoming signal is routed directly to an A/D converter. The resulting data bytes are stored in RAM, used to plot an image on the CRT screen, and stored directly on disk.

The best digital oscilloscopes handle signals in the range of only 10 MHz, and also run into trouble with limits on available memory. At a sampling rate of 10 MHz, you get 64K data bytes in just 64 milliseconds. Nonetheless, for brief or "transient" waveforms in this frequency range, a digital scope offers substantial benefits.

If you put an A/D converter into an Apple, you have the makings of a digital storage oscilloscope. RC Electronics and Northwest Instruments both acted on this fact by assembling the necessary extra ingredients to provide most of the amenities of a standard oscilloscope system (trigger inputs, BNC connectors, volts/div and sec/div controls, etc.). In fact, the products from the two companies are not really comparable. The Northwest Instruments Model 85 aScope is nothing more than a rather slow eight bit A/D converter with 256 bytes of memory available for storage. It handles its 256 data points at up to 10,000 points per second, which is similar to the data flow in some of the weaker A/D conversion software packages.

The Model 85 does have a trigger feature which lets it do "equivalent time" digitization of signals at up to 50 MHz. This means that if you have a repetitive signal source which produces exactly the same waveform every time, you can set up the Model 85 to sample the signal many hundreds of times until its actual shape has been digitized. Software for this system is not extensive, and the performance is far lower than the RC Electronics systems.

The three Applescope products from RC Electronics are actually formidable digital storage oscilloscopes. Despite the far more sophisticated advertisements for the Model 85, it is the Applescopes which will be useful for most interested labs. The APL-D2 is based on an eight bit tracking A/D converter which can follow one MHz components in some waveforms. The HS-7 is based on a seven bit "flash" converter with a conversion time of 70 nanoseconds (14 MHz), and the HR-14 is based on a hybrid ADC which uses both flash and tracking A/D conversion to get 14 bit precision at 500 kHz.

The Applescopes benefit from fast machine language software and from on card memory buffers which circumvent the limits imposed by the processing speed of the 6502. For sampling rates up to 28 kHz, the scope driver software can load data directly into a 16K buffer in the Apple. For higher sampling rates, the data is sent first to a RAM buffer on one of the Applescope cards. The APL-D2 has a 1K buffer, but the HR-14 and HS-7 have 2K buffers. These buffers

can store points on up to the full maximum sampling rate of 14 MHz in the HS-7. Once a waveform is digitized into the buffer, the scope driver software can transfer it on down into the Apple's main memory a few milliseconds later. (Although the literature from RC Electronics claims a DMA capability (see Chapter 27), the Applescope boards do not actually do DMA.)

In evaluating the acquisition speed needed for a given waveform, you should keep in mind the peculiarities of the ADC method used in these systems. As noted earlier, a "tracking" ADC can follow small changes rapidly, doing conversion at speeds approaching the stepping rate. However, some kinds of signal changes make for "worst case" performance and may require many steps to complete conversion.

The APL-D2 can step its control register at 0.286 microsecond intervals, and it uses a modified tracking algorithm for handling large jumps, but the worst case conversion time includes 10 steps or 2.6 microseconds for acquisition and 6.6 microseconds for settling. Thus the worst case conversion time is a little more than nine microseconds; this means a 100 kHz sampling rate for some kinds of signals, but up to 3.5 MHz for others. The HR-14 has a slowermaximum sampling rate (500 kHz), but it has a faster settling time and seven bits of flash conversion, so it may have a faster effective sampling rate than the APL-D2 for some kinds of signals.

The final point to keep in mind is that the conversion speed must be faster than the frequency of the signal being analyzed in order to get an accurate description of the waveform. The Nyquist Limit suggests that you need a sampling rate at least twice as fast as the frequency of the waveform. To begin with, to even approach this limit you need a very good "track and hold amplifier" which helps you avoid averaging the signal while it is being sampled. In practice, however, you shouldn't expect to use a sampling rate much less than three times the signal frequency, and some waveforms require an even larger margin for accurate digitization. Given the ambiguities about worst case conversion speed for a tracking ADC, and about appropriate sampling rate, the best thing is to contact the engineers at RC Electronics to discuss your particular application.

Data Acquisition and Process Control

Microcomputers have been revolutionizing the scientific and industrial workplace. While much of the excitement about microcomputers has focused on business and database oriented tasks, the advent of inexpensive smart instrumentation and control systems has been having a tremendous impact on productivity in the laboratory and in production facilities.

A great deal of the impact has been based on individual scientists and technicians who have engineered interfaces for their own applications, but the demand for these systems has grown so rapidly that it is now possible to purchase fully engineered and assembled systems which can handle a remarkable variety of tasks.

Three important system manufacturers are Keithley/DAS, Cyborg, and Interactive Structures. The products from these three companies each have a distinct emphasis. The Series 500 from Keithly/DAS has a remarkable array of input preamplifiers and output shaping interfaces any of which can be installed in a single 10 slot external chassis. This system also emphasizes rapid solid state switching among hundreds of isolated inputs and outputs. It is possible to attach more than one Series 500 chassis to an Apple, but a very diverse system with hundreds of channels can be assembled in a single chassis.

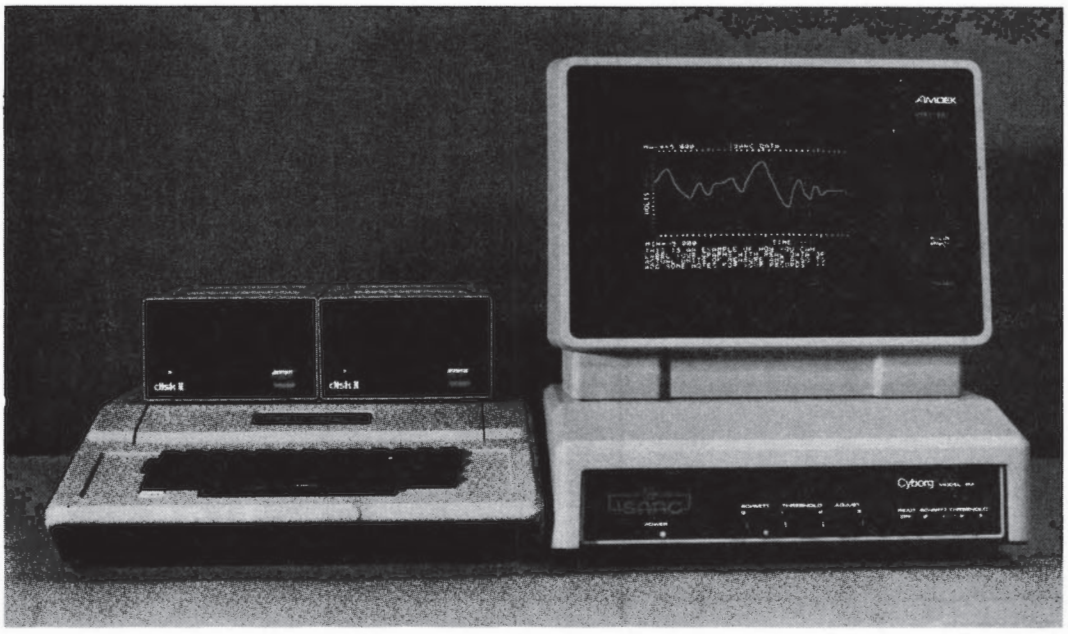


Fig. 15.3 ISAAC 91A data acquisition system.

The Cyborg products can be organized into several configurations which have strong input capabilities but which do not match the Keithley/DAS output features. The ISAAC 41A/91A (see Figure 15.3) core system provides for selection among a variety of high resolution A/D converters and other interfaces in eight slots. You can expand it to include a variety of preamplifers, but this requires the addition of a special "remote" chassis and the signals are multiplexed by comparatively slow reed relay switches. In addition, Cyborg has a new "smart chassis" called the ISAAC 2000 which uses a 68000 microprocessor and up to two megabytes of RAM to sock away huge numbers of data points collected at high speed on one to four channels.

The DAISI system from Interactive Structures features a compact, rugged set of interfaces many of which can be installed directly in Apple slots without an external chassis, but which can also be purchased from Eco-Tech which makes rack mount chassis systems based on the DAISI cards. The DAISI system has the unique distinction of being the first microcomputer based data acquisition system to be used in Earth orbit aboard the Spacelab, thus passing muster at NASA, surviving launch and reentry in the Space Shuttle, and working flawlessly for a week in space. This system may be the best choice when it is convenient to spend $2,500 on each of five Apples each with its own partial DAISI systems scattered around a lab rather than, for instance, to spend $12,500 on a single ISAAC 2000 system.

Elements of a System

To compare the three systems, you can group the various features they provide into seven categories: analog signal inputs, analog to digital conversion, digital to analog conversion, digital I/O, clock/counters, power control and monitoring, and physical layout (see Table 15.2). In addition, it is important to consider the quality of the software tools available for organizing the whole ensemble. The software varies with regards to integration of graphics output, and with regard to speed of data handling.

	A/D Conv.				D/A Conv.			Dig. I/O		Timer	Iso Amp	Relays		Real Time Clock
	8 bit	12 bit	14 bit	16 bit	8 bit	12 bit	16 bit	8 bit	16 bit	16 bit		opto	E/M	
Applied Eng.	8	8												
CCS 7440										3				
Data Translation		16	16	16										
Eco-Tech	16	16			8			4		4				
Hollywood								6						
IMI		(8)	1			(8)		2		2		✓		
Interactive Str.	16	16			8			4		4		✓	✓	
ISAAC		16 2		2		4			2		✓			✓
Keithley DAS		16 or 272	16 or 272			5	2		11, 1o	2 (5)	✓	✓	✓	✓
uD-1000	8							11, 1o						
Mountain	16				16									
Northwest Inst.	1													
RC Electronics	2		1											
Snave								2		2				

Table 15.2 Lab system features. Number of channels available is noted in grid.

Organization of Analog Input and A/D Conversion

Both the ISAAC and the DAISI analog input sections are based on multichannel A/D converters, principally the ISAAC I-100 and the DAISI AI13. In most configurations, analog signals are connected directly to the A/D converter card, ranges are set, and conversion can proceed. The DAISI system offers a standard "Signal Conditioner" which can be used as a preamp, but many acquisition systems based on the DAISI AI13 or AI02 will need external isolation amplifiers, etc., built by the user or purchased from another source. Channel selection takes place within the A/D converter chip in both systems. For high precision tasks, the ISAAC I-150 is distinguished as the only 16 bit A/D converter available among the systems.

The Keithley/DAS system (see Figure 15.4) is based on one A/D converter with just a single channel of input. A system can be configured either with the 12 bit ADM1 or with the 14 bit ADM2. All of the multiplexing and signal conditioning is done on separate analog input cards before the conditioned signal is passed along to the A/D converter. The AIM1 through AIM6 modules include a variety of top quality instrumentation amplifiers and isolation amplifiers with configurations for inputs ranging from 5 mV to 10 Volts from strain gauges, thermocouples (B, E, J, K, R, S, and T with cold junction reference), RTD's, and just about any other kind of sensor or source that might turn up. The multiplexing is handled by high speed CMOS switches.

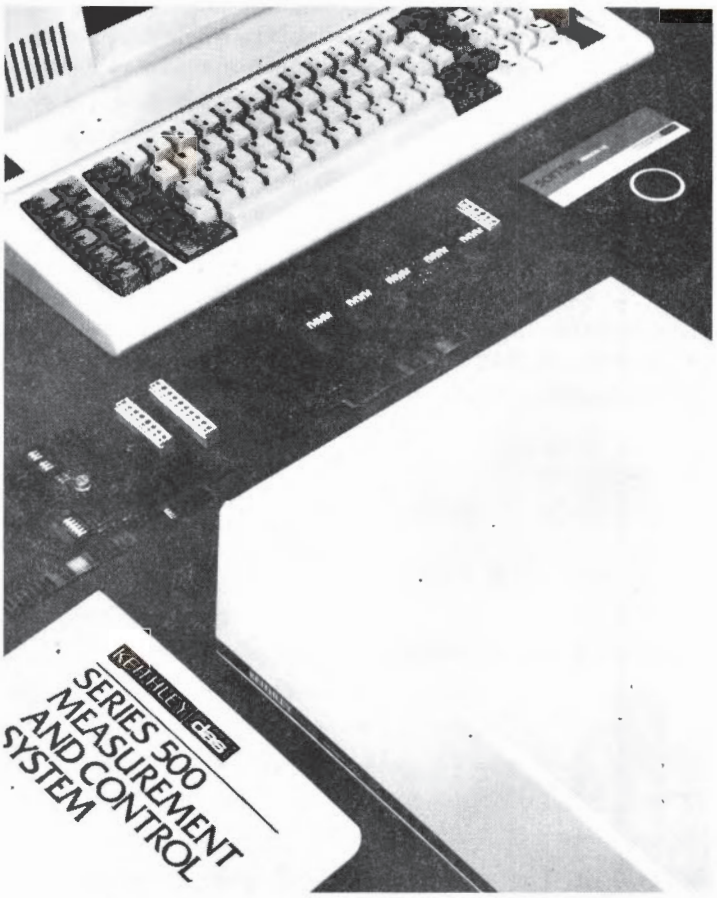


Fig. 15.4 Series 500 data acquistion system.

You can set up a remote preamplifier system for the ISAAC products as well, but this is a higher cost, lower performance proposition. To set up two of the 4 channel I-140 preamps, you need an additional remote chassis, an I-130 card to interface the remote system with the main ISAAC bus, and an I-160 multiplexer, the three of which cost $1,450 before you even choose the preamp cards, each of which costs another $750. Thus, adding eight channels of thermocouple preamps costs about $3,000 for the ISAAC systems, but only a few hundred dollars in the Keithley/DAS system and you can only use J, K and T thermocouples in the standard ISAAC configuration. Further, the multiplexing is done by reed relays with no debouncing, so it takes nearly half a second to switch channels.

Software Speed for Storing Data Points

For the Keithley/DAS, DAISI, and ISAAC 91A/41A systems, the actual rate of data collection is limited by the speed at which a machine language program running on the 6502 can collect in the data points and store them. For single channel data acquisition, the Keithley/DAS software provides the fastest data collection, running up to 27 kHz, which is roughly equivalent to a free running A/D converter with a 37 microsecond conversion time. Most of the software for the DAISI and ISAAC systems has a maximum speed of 10 kHz, but both offer special machine language routines to push the speed up to 20 kHz.

An important feature of the Keithley/DAS and DAISI software which is missing from the ISAAC system is "foreground/background operation," which means that the Apple can proceed with statistical analysis and plotting simultaneously with management of slower (500 Hz) data collection tasks. The implementation of software multitasking on the Apple II with DOS 3.3 is no simple feat (see above).

The ISAAC 2000 system (see Figure 15.5) far exceeds the data collection speeds of any other system. The separate 8 MHz 68000 processor in the ISAAC 2000 can handle sampling rates up to 200 kHz for a single channel and up to 50 kHz for four channels running simultaneously. Since 200 kHz is equivalent to an A/D conversion time of five microseconds, Cyborg has built its C-100 module around a fast converter chip that runs at that speed. The system also takes advantage of the huge linear address space of the 68000 microprocessor (see Chapter 30) by allowing the addition of two Megabytes of RAM. The resulting potential for 10 seconds of real time digitization at 200 kHz is a very impressive feat for a microcomputer based system.

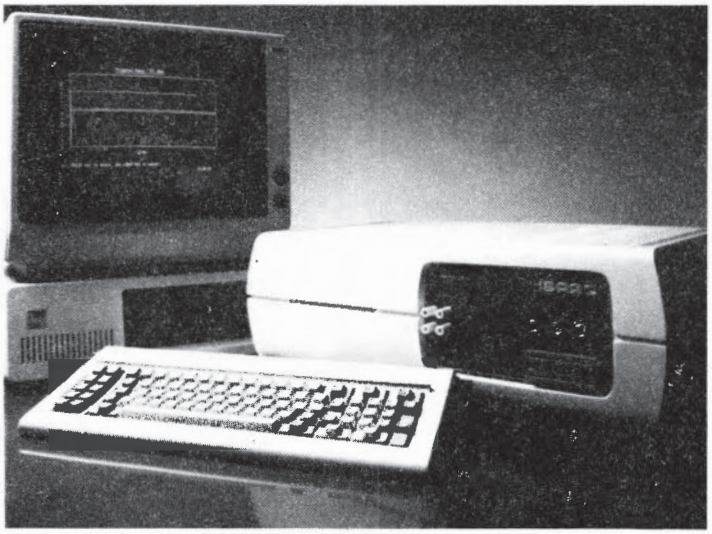


Fig. 15.5 ISAAC 2000 data acquistion system.

Digital I/O and Timing

The DI09 board in the DAISI system is based on two 6522 VIAs (see Chapter 14), thus it provides four bit programmable, bidirectional, eight bit parallel ports with handshaking. There is no special provision for buffering or isolating the inputs, and the outputs can only drive a single TTL load. Each of the two 6522s has two 16 bit programmable counter/timers which can also be used for event counting and as programmed frequency generators and delay timers.

The ISAAC 41A/91A system provides a real time clock on the interface card in the Apple, and some counters in the 91A chassis. The digital I/O is handled separately by the I-120 module which has 16 bits of input and 16 bits of output. Each of these ports can be programmed to operate on a bit, byte, or word (16 bit) basis, and the outputs can drive considerable loads. The ISAAC 2000 includes a 24 bit timer/counter but no real time clock. Digital I/O for the ISAAC 2000 is also based on the I-120.

These various functions are spread onto several boards in the Keithley/DAS system. The real time clock is on the interface card in the Apple, and, like the ISAAC clock, has a resolution of one second. Digital I/O requires two separate boards, but the inputs are optically isolated

(see Chapter 14) and have variable resistors to adjust for the strange parallel output voltages that always seem to turn up on older lab instruments. The PIM1 is a powerful programmable counter/timer with five 16 bit registers which can be cascaded into a single 80 bit register.

Analog Outputs and Power Control

Just as with analog and digital input, the Keithley/DAS system has the greatest variety of options for analog output and control. The DAISI analog output is based on a two channel, eight bit D/A converter with a five milliAmp output, and the ISAAC I-110 offers a four channel 12 bit D/A converter, also with a five mA output. The Keithley/DAS has three different A/D converter boards, with the AOM1 being similar to the ISAAC I-110, the AOM2 offering two channels of 16 bit precision, and the AOM3 emphasizing high current tasks with four separate 10 bit D/A converters each capable of 20 mA output.

The ISAAC system has no provision for power control and monitoring, which does happen to be a strong point of the DAISI system. You get a separate chassis in which you mount your choice of optically isolated relays for switching 110 VAC and 60 VDC or power monitors with similar voltage ranges. The relays and monitors are supervised by signals from the DAISI digital interface board. The Keithley/DAS power control system is a little bit more elegant in that it has greater voltage ranges, uses solid state relay subsystems, and does not require an additional chassis.

Physical Layout

The Keithley/DAS modules all plug into a standard 10 slot chassis with a heavy duty transformer based power supply (see Figure 15.4). This makes for a compact system which is convenient to expand or alter. The ISAAC systems involve six different chassis types all for handling different sizes, shapes, and numbers of cards. This is not a reflection of bad planning so much as it is a reflection of evolutionary growth of Cyborg systems over the past six years (there are nearly 1,500 ISAAC systems in use in laboratories all over the world).

The way to approach the ISAAC configurations is to begin with the 91A chassis (see Figure 15.3). It includes all of the features of the I-100, I-110, and I-120 boards (see Table 15.1) as built-in features. In addition, there are eight slots into which additional I-100, I-110, I-120, I-150, I-180 and I-130 cards can be installed. The 41A chassis has got just four slots and no built-in features. If you need to use an I-160 multiplexer and/or I-140 preamplifier, you have to get a different chassis because these cards are physically larger than the 91A/41A cabinets. Cyborg offers a two slot and a five slot chassis for these larger boards.

Things get even more complex with the ISAAC 2000 (see Figure 15.5). The main chassis has the 68000, 128K of RAM, 32K of ROM, a 24 bit timer, an IEEE-488 port (see Chapter 20), two RS-232C ports (see Chapter 16), two slots for "C" cards, four slots for "I" cards, and ports for attaching other chassis'.You'll need one of the C-slots for the fast A/D converter, and the C-200 board has just 256K of RAM, so you need an extra chassis with six more C-slots to get up to two megabytes of RAM. Then, you might choose to attach one 91A or four 41A chassis', and then to extend on out to some remote chassis' for preamps.

Aside from the complexity of all this, you spend a lot of money on boxes, and all this ribbon cable can't be too good for signal integrity at high speed. A Cyborg system which fits entirely within a 91A or a 2000 chassis gives you good performance for your money, but, especially if you're going to need more "I" slots or preamps, you might do better to start off with a Keithley/DAS.

The DAISI cards (AI13, AI02, DI09, AO03) all fit in regular Apple slots (see Figure 15.6), but you need to buy a chassis for the signal conditioners and/or for the power control and monitoring equipment. If your Apple runs out of slots or the power supply gets overtaxed, you'll need more Apple slots. If you have fairly good skills as an Apple programmer, you can purchase a Mountain Expansion Chassis (see Chapter 22) which gets you eight more slots and a good power supply.

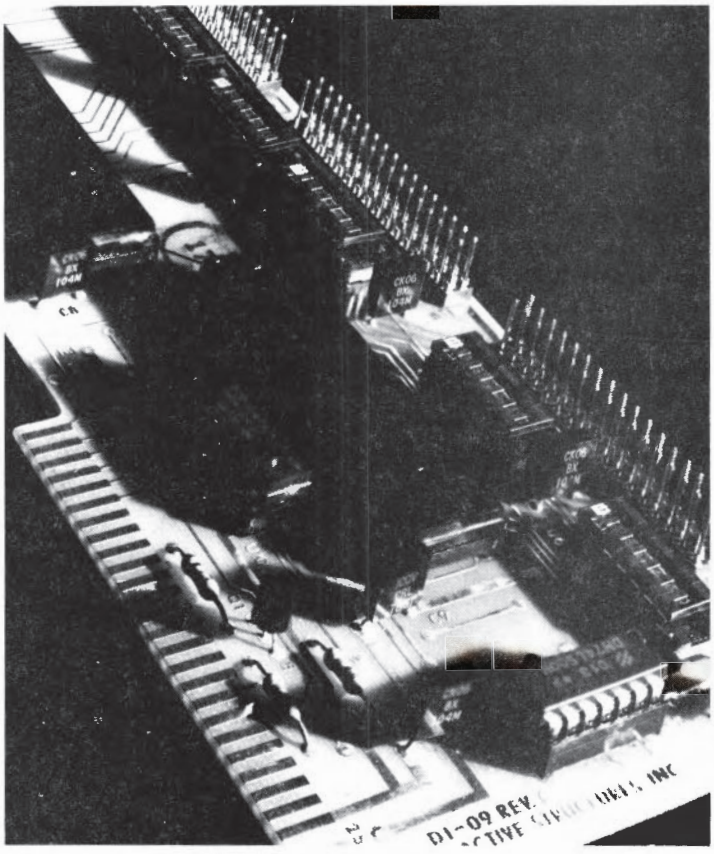


Fig. 15.6 DAISI DI09 digital I/O card with two 6522 VIAs.

Many users, however, may want to consider buying an ALIS system from Eco-Tech which is effectively the same thing as buying an expansion chassis based on a DAISI card except that Eco-Tech has done all the hardware and software work for you. This may be a good idea if the DAISI electronics meets your needs but you don't want to get involved in all the details of interfacing. If you're starting from scratch and contemplating a large system, you should probably give Keithley/DAS consideration first since it's much more economical.

Software Support for Programmers and Users

All of these systems feature simple extensions to Applesoft which permit easy programming. The programming tools from Keithley/DAS go in for helping you pull off such feats as 16 inputs, in color, scrolling sideways on the screen or simultaneous statistical analysis during data collection. One advantage of ISAAC, on the other hand, is that there have been so many systems in use for so long that there are a number of "turnkey" software packages for certain applications which avoid the need for programming altogether.

Specialized Systems for Molecular Biology and Biochemistry

In the past, automated chemical analysis has largely been the province of well financed medical facilities; however, there are now a variety of tasks in which research biologists and chemists can economically make good use of microcomputers. There are commercially available hardware and software packages for gel scanning, peptide and DNA synthesis, luminescence assay, centrifuge control, and chromatography.

The chromatography monitoring and control systems are in such widespread use that they are manufactured by third party sources such as Interactive Microware and Cyborg, while the other systems are specialized packages developed by the manufacturers of the analytic instruments. In most cases, the Apple replaces the strip chart recorder and/or teletype (see Chapter 13, Figure 13.2), but some systems let you program the timed or triggered operation of valves, switches and other controls.

Gel Scanning and Luminometry

There are two packages from LKB which use an Apple for data collection and analysis. These include an interface for their 2202 Ultroscan Laser Densitometer and for their 1250 Luminometer. The densitometer package is used as an extension of their high resolution electrofocusing system, but the densitometer can be configured to read polyacrylamide, agarose, and cellulose acetate gels, tube gels, micro TLC plates and autoradiography films, all with a resolution of up to 50 microns, so the LKB Gel Scan software and/or hardware system may be a convenient starting point for automating the analysis of a wide variety of separations.

The package for the luminometer is based on a "luminescence immuno assay" which competes with a standard RIA. LKB markets this system in conjuction with chemiluminescent reagents for steroid detection. The points to keep in mind about luminescence are that it can be more sensitive than scintillation counting, but that ATP based luminescence generates a flash over a period of just a few seconds and in which the shape of the waveform is often important. Therefore, the proper monitoring of ATP bioluminescence as well as slower FMNH/NADH bioluminescence is more demanding on the computational power of the photometer system than standard counting of isotopes.

Many scintillation counters have a direct output from the photometer so you can do pulse counting for luminescence assays directly in an Apple with a timer/counter interface card (see Chapter 14), thus bypassing the scintillation counting system. However, the LKB package is a straightforward way of getting a complete system.

Automated Synthesis and Separation of Macromolecules

While full scale automation of the synthesis and purification of macromolecules is nowhere in sight, there are a few steps you can take to improve productivity along these lines. Vega Biotechnologies offers a monitoring and control interface called the Coder 280 which links an Apple and an automated DNA synthesizer for producing oligomers. Their other Apple-based system is the Coupler 1000 for peptide synthesis. In both of these systems, using an Apple means that you can run 24 hours a day if necessary, but that your investment in microprocessor control also gets you a standard microcomputer to use when you're not running syntheses.

Beckman has configured their L8M ultracentrifuge for a control and spin monitoring interface with an Apple. This not only simplifies your task in calculating the speed for different rotors, but it also means more unattended operation, temperature control, selection of stored acceleration/deceleration profiles for gradient separations, etc.

Chromatography Systems

Both IMI and Cyborg make hardware and software packages for monitoring High Pressure Liquid Chromatography (HPLC) and Gas Chromatography (GC). These systems are based on comparatively inexpensive slow A/D converters but can be expanded to include control over gradients, etc. The IMI Adalab system uses a 12 bit, dual slope converter with a 50 millisecond conversion time. This is an adequate sampling speed for standard column HPLC, but may not be fast enough for newer high speed separation techniques.

The Cyborg ISAAC 42A is built around an SAR A/D converter (see Chapter 14) with a 10 millisecond conversion time and 16 bit precision. The higher speed of conversion makes this system suitable for microbore HPLC and capillary GC. The 16 bit precision is something to keep in mind if you're doing complex separations with trace quantities. A 12 bit converter effectively divides the full scale into 4,000 steps while a 16 bit converter divides it into 65,000 steps. Whether or not this makes a difference depends on the quality of your detector; if you've spent the money for a very high resolution separation and detection system the 16 bit precision may be well worth the extra cost in the data collection system.

Both of these systems have fairly extensive ready to run chromatography data collection and analysis software. The Chromatext package with the ISAAC 42A has more elaborate analysis and graphics features, and uses a 294K RAM card for high speed data storage and analysis tasks. However, it is missing two strong features from the IMI package: software control of the gradient pumps, and built-in routines for operating an HP 7470A/7475A or Houston Instruments DMP 29 plotter (see Chapter 19).

To take advantage of IMI's software support of control outputs, you need to get their Chromadapt interface module along with the basic Adalab card. The ISAAC 42A chassis from Cyborg has four slots for interface modules, much like their ISAAC 41A chassis, but it is sold with just the A/D card installed and the other three slots empty in the standard chromatography package. If you want to add output control to the ISAAC system you have to purchase an additonal card and their LabSoft programming language.

PART 4

The Connected Apple

- **CHAPTER 16 Serial Signals**
- **CHAPTER 17 Modems**
- **CHAPTER 18 Interfaces for Terminals, Modems and Printers**
- **CHAPTER 19 Printers and Plotters**
- **CHAPTER 20 Local Area Networks and IEEE-488**


TELETYPE

Chapter 16

Serial Signals

In the dawning decades of the electronic age, generation after generation of telegraph operators struggled to make out the dots and dashes of Morse code. This long era came to an end in the first few years of the 1900s with the advent of the printing telegraph, a machine which could carry out the task of detecting and decoding, and then typing out the received message as universally recognizable alphabetic characters.

From that time, the transmission and reception of electronically coded information ceased to be a realm dominated by human skill, training and experience, and instead, the focus turned to three areas: designing a code system which best suited the interpretive abilities of available machines, choosing the appropriate electrical characteristics to best enhance reliable transmission of the coded signals, and physically connecting the right wires to the right places.

It is an interesting sort of footnote that we still teach Morse code to youngsters (Boy Scout merit badges and the like). This is apparently done as a sort of ethnic inculcation drawn from the traditions of the dimly remembered past in which our electronic culture was born. More recently, "RS-232C" has replaced Morse code telegraphy as a focus of awe and mystery.

Moving ASCII codes

For the past 20 years or so there has been substantially universal acceptance of the ASCII code as a way of representing numbers and letters for interpretation by machines. The ASCII code was discussed in Chapter 2, it will be discussed in more detail in Chapter 18, and it is shown in Chapter 18, Table 18.1. Its most recent competitors included Binary Coded Decimal (BCD), and Extended Binary Coded Decimal Interchange Code (EBCDIC).

BCD still crops up all the time inside computers, but is almost never permitted to get out onto communications lines. For many years, all IBM computers used EBCDIC exclusively. Fortunately, the ASCII world came to dominate, and EBCDIC has disappeared so completely that you can reasonably expect never to have to deal with it outside of discussions of computer history.

This is not to imply that you can simply plug any piece of computing equipment into any other piece and expect them to communicate instantly and without further fuss. Far from it. Far, far from it.

The code itself is never the problem. What has happened is that the functions performed by the printing telegraph have been distributed out into several different kinds of machines. ASCII codes come from people working at CRT terminals or microcomputers, and from data-

bases in huge mainframe computers. The codes get sent to modems which place them onto the telephone lines, to printers, and to clusters of other nearby computers. Modems, telephone lines, printers, terminals and computers all seem to want their ASCII codes served up in their own particular favorite way.

Connecting to Modems and Printers

There are no absolute and general standards. Variety is legion. In Chapter 20 there is coverage of special interfaces for multi-computer networks, for connections with lab instruments (IEEE 488), and for talking with old teletypes. However, most Apple owners can limit their concerns to interfacing one kind of "terminal" (the Apple) to a modem and to a printer.

Connection between terminals and modems is the one completely standardized kind of interface. The coding system is called "asynchronous serial," the electrical standard is called "RS-232C," and the arrangement of wires in the connector is also called "RS-232C."

Unfortunately, the RS-232C system was designed only for operating modems, and happens to lack two signals which you must have to operate a printer. As a result, nearly every printer manufacturer has invented his or her own type of connection system. These systems fall into two broad categories. The first group tries to come as close to RS-232C as they can, however they tend to differ with regard to their way of adding the two missing signals.

The second group of printer manufacturers uses a set of slightly different coding systems collectively called "parallel," an electrical standard called "TTL," and several different kinds of cable connectors. Within the last year or two, nearly all printers have been sold with the option of using a standard system of wire connections and code signals. This de facto standard is called "Centronics Parallel" and has been imposed on the manufacturers by frustrated consumers rather than by an established standards committee.

Interface Cards and //c Ports

The II, II+ and //e have no built-in facilities for any kind of RS-232C or parallel connections. As a result, the usual pattern has been to select your printer, and then to choose an expansion card which has the appropriate kind of modified RS-232C or specialized parallel connections, and possibly to buy a separate standard serial card for your modem. None of the printer manufacturers makes interface cards and vice versa, so you have to match up equipment from at least two different companies.

Modem manufacturers have been a bit more attuned to helping folks with interfacing. SSM sells a serial interface card and an external modem, while Hayes, Multi-Tech, SSM and Novation sell systems in which the guts of the modem is actually plugged directly into an Apple peripheral slot, eliminating the need for a serial interface card.

One safe route through this jungle is to buy a card called the AIO-II from SSM Microcomputer Products. This one card can be simply and conveniently used to connect an Apple to absolutely any printer, CRT terminal, or external modem (as well as to most plotters and a few other specialized kinds of equipment). SSM will also provide any kind of interface cable you could possibly need. This card is a bit of an oddity in the microcomputer world because it's inexpensive, it's always compatible, and no one else makes one.

The situation is much more straightforward in the //c. The machine has two built-in, high performance RS-232C ports and no parallel ports. Therefore, printer and modem manufacturers know that they must adjust their equipment to work with the //c. The parts and firmware in the //c are fairly sophisticated and flexible, so you are unlikely to run into any insurmountable difficulties. Further, since Apple now sells its own modems and printers, you can avoid all the problems by getting everything directly from Apple.

Computer Communication Via Telephone Lines

When data is being moved about within a computer, it is reasonably safe to assume that the bits placed on the data bus by the 6502 will arrive, for instance, at the RAM chips exactly as intended. Further, it is safe to assume that the RAM chips will be perfectly synchronized with the 6502's actions and that they will respond by capturing the data, without fuss, under simple direct order from the 6502.

None of these assumptions are reasonable or safe when data bits depart from within the protected confines of the computer. Inside the computer, signals are protected by a variety of electronic arcania such as "thick ground planes," "bypass capacitors," and "power line filters" (see Chapter 13), and all are synchronized by direct, "hard wire" connections to the system's one master clock.

Outside the computer, signals may travel through unprotected wires with noise prone electrical characteristics. Electronic noise from radio frequency signals and from other signal wires can and often will cause bits to get changed during transmission. Once the data arrives at its destination, the receiving chips may belong to a computer that operates at a completely different speed than the sender. Even if the two parties to the communication are both Apples, their clocks will certainly not be synchronized with each other, and there is no way to make the master clocks of two Apples become synchronized.

The result of these problems with noise and timing is that the basic information represented in ASCII code must be supplemented with additional coded information which can help maintain surveillance against errors and which can help with the synchronization problem.

The Asynchronous Serial Coding System

The fundamental logic behind the various communications coding systems is that there can be two completely different kinds of bits in the coded signals which are sent from one machine to another. Some bits will be assigned to carrying the data which is actually being sent, but there will also be some bits which are assigned exclusively to the special needs of the communications equipment. These communications bits are usually added to the data just before it is sent out of the computer, and then stripped off by the receiving device after the signal is captured but just before the data is presented to the receiving processor.

To be historically accurate it is worth noting that the old serial interface card sold by Apple relied on a system in which the 6502 itself added or removed the communications bits. However, this kind of work has long been relegated to specialized single chip communications controllers. The two most popular of these for Apple cards are the 6850 from Motorola and the 6551 from MOS Technology, Synertek and others. These chips for 6800 based or 6502 based computers are both called Asynchronous Communications Interface Adapters (ACIAs). The 6850 turns up in the AIO-II card, while the newer and slightly more versatile 6551 is at the heart of the

new Super Serial Card (SSC) from Apple. The //c has two Synertek 6551s for its two ports. Similar parts in 8080 or Z-80 based systems are called Universal Asynchronous Receiver Transmitters (UARTs).

In asynchronous communications, each individual character which is transmitted is treated as a separate problem for the timing and data integrity system. In other systems, long streams of characters are sent with communications bits attached only to the very beginning and very end of the stream. However, in the standard form of asynchronous communications, there are three bits of communications information for every seven bits of data.

Error Checking and Parity

The ASCII code itself represents each of its 128 characters and control codes as a seven bit pattern (see Chapter 18, Table 18.1). Therefore, asynchronous coding systems assume that when ASCII data is being transmitted, the eighth bit in each byte is unused. This eighth bit is manipulated by the ACIA to provide a crude sort of error detection system. The ACIA examines each ASCII code and treats it as if it were just a number between zero and 127. In one common protocol, it sets the eighth bit to zero if the number is even, but puts a one in the eighth position if the number is odd. This system is called "parity checking."

To use this for error checking, the receiving ACIA must be set up to operate with the same protocol. When it is, it will check the numeric value of the incoming ASCII codes, and then check to be sure the eighth bit is set appropriately. The five major error checking variants available from the 6850 and 6551 (see Figure 16.1d) are odd parity, even parity, mark parity (in which the eighth bit is set to one on all characters, a system expected by many mainframe computers), space parity (in which the eighth bit is set to zero), and no parity. This last option is important if you want to transmit non-ASCII data such as binary files or printer graphics commands which use all eight bits.

If the ACIA is not happy with what it finds in the eighth bit, it is able to report that this character is certainly incorrect. A program which is operating an ACIA should always check with the "status register" of the ACIA (see Figure 16.1e) to learn if it thinks a character is bad.

The asynchronous communication system provides no mechanism for correcting the error, but at least you've got a warning that bad data is coming in. When this happens in a text transmission, you can simply examine the text and correct it. If it happens with numerical data, this sort of warning is very important. Occasional errors are handled by repeating the offending portion of the transmission. If large numbers of errors are coming in, there's probably some hardware problem (for example, interference in the telephone line) which you're going to have to find and correct before continuing.

Asynchronous Communication Interface Adapter

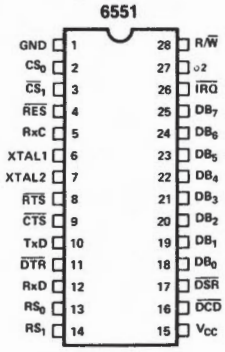
Features

- On-chip baud rate generator: 15 programmable baud rates derived from a standard 1.8432 MHz external crystal (50 to 19,200 baud).
- Programmable interrupt and status register to simplify software design.
- Single +5 volt power supply.
- Serial echo mode.
- False start bit detection.
- 8-bit bi-directional data bus for direct communication with the microprocessor.
- External 16x clock input for non-standard baud rates (up to 125 Kbaud).
- Programmable: word lengths; number of stop bits; and parity bit generation and detection.
- Data set and modem control signals provided.
- Parity: (odd, even, none, mark, space).
- Full-duplex or half-duplex operation.
- 5, 6, 7, 8 and 9 bit transmission.

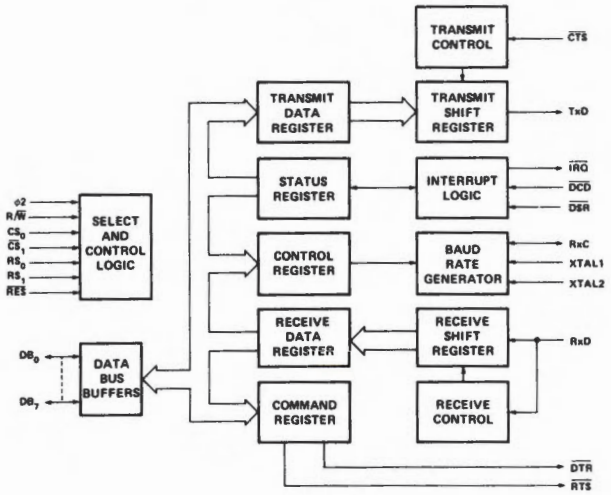
Description

The SY6551 is an Asynchronous Communication Adapter (ACIA) intended to provide for interfacing the 6500/6800 microprocessor families to serial communication data sets and modems. A unique feature is the inclusion of an on-chip programmable baud rate generator, with a crystal being the only external component required.

Pin Configuration



Block Diagram



Interface Signal Description

$\overline{RES}$ (Reset)

During system initialization a low on the $\overline{RES}$ input will cause internal registers to be cleared.

φ2 (Input Clock)

The input clock is the system φ2 clock and is used to trigger all data transfers between the system microprocessor and the SY6551.

R/$\overline{W}$ (Read/Write)

The R/$\overline{W}$ is generated by the microprocessor and is used to control the direction of data transfers. A high on the R/$\overline{W}$ pin allows the processor to read the data supplied by the SY6551. A low on the R/$\overline{W}$ pin allows a write to the SY6551.

$\overline{IRQ}$ (Interrupt Request)

The $\overline{IRQ}$ pin is an interrupt signal from the interrupt control logic. It is an open drain output, permitting several devices to be connected to the common $\overline{IRQ}$ microprocessor input. Normally a high level, $\overline{IRQ}$ goes low when an interrupt occurs.

DB_0 - DB_7 (Data Bus)

The DB_0-DB_7 pins are the eight data lines used for transfer of data between the processor and the SY6551. These lines are bi-directional and are normally high-impedance except during Read cycles when selected.

CS_0, $\overline{CS_1}$ (Chip Selects)

The two chip select inputs are normally connected to the processor address lines either directly or through decoders. The SY6551 is selected when CS_0 is high and $\overline{CS_1}$ is low.

RS_ϕ, RS_1 (Register Selects)

The two register select lines are normally connected to the processor address lines to allow the processor to select the various SY6551 internal registers. The following table indicates the internal register select coding:

Fig. 16.1a Pinout, block diagram, and microcomputer interface signals for Synertek 6551 ACIA (Asynchronous Communications Interface Adapter). The Transmit Data Register and the Receive Data Register both appear in the Apple //c address space at $C098 for the port 1 ACIA or at $C0a8 for the port 2 ACIA.

Courtesy of Synertek, see page 2.

RS_1	RS_0	Write	Read
0	0	Transmit Data Register	Receiver Data Register
0	1	Programmed Reset (Data is "Don't Care")	Status Register
1	0	Command Register	
1	1	Control Register	

The table shows that only the Command and Control registers are read/write. The Programmed Reset operation does not cause any data transfer, but is used to clear the SY6551 registers. The Programmed Reset is slightly different from the Hardware Reset ($\overline{RES}$) and these differences are described in the individual register definitions.

ACIA/Modem Interface Signal Description

XTAL1, XTAL2 (Crystal Pins)

These pins are normally directly connected to the external crystal (1.8432 MHz) used to derive the various baud rates. Alternatively, an externally generated clock may be used to drive the XTAL1 pin, in which case the XTAL2 pin must float.

TxD (Transmit Data)

The TxD output line is used to transfer serial NRZ (non-return-to-zero) data to the modem. The LSB (least significant bit) of the Transmit Data Register is the first data bit transmitted and the rate of data transmission is determined by the baud rate selected.

RxD (Receive Data)

The RxD input line is used to transfer serial NRZ data into the ACIA from the modem, LSB first. The receiver data rate is either the programmed baud rate or the rate of an externally generated receiver clock. This selection is made by programming the Control Register.

RxC (Receive Clock)

The RxC is a bi-directional pin which serves as either the receiver 16x clock input or the receiver 16x clock output. The latter mode results if the internal baud rate generator is selected for receiver data clocking.

$\overline{RTS}$ (Request to Send)

The $\overline{RTS}$ output pin is used to control the modem from the processor. The state of the $\overline{RTS}$ pin is determined by the contents of the Command Register.

$\overline{CTS}$ (Clear to Send)

The $\overline{CTS}$ input pin is used to control the transmitter operation. The enable state is with $\overline{CTS}$ low. The transmitter is automatically disabled if $\overline{CTS}$ is high.

$\overline{DTR}$ (Data Terminal Ready)

This output pin is used to indicate the status of the SY6551 to the modem. A low on $\overline{DTR}$ indicates the SY6551 is enabled and a high indicates it is disabled. The processor controls this pin via bit 0 of the Command Register.

$\overline{DSR}$ (Data Set Ready)

The $\overline{DSR}$ input pin is used to indicate to the SY6551 the status of the modem. A low indicates the "ready" state and a high, "not-ready." $\overline{DSR}$ is a high-impedance input and must not be a no-connect. If unused, it should be driven high or low, but not switched.

Note: If Command Register Bit 0 = 1 and a change of state on $\overline{DSR}$ occurs, $\overline{IRQ}$ will be set, and Status Register Bit 6 will reflect the new level. The state of $\overline{DSR}$ does not affect either Transmitter or Receiver operation.

$\overline{DCD}$ (Data Carrier Detect)

The $\overline{DCD}$ input pin is used to indicate to the SY6551 the status of the carrier-detect output of the modem. A low indicates that the modem carrier signal is present and a high, that it is not. $\overline{DCD}$, like $\overline{DSR}$, is a high-impedance input and must not be a no-connect.

Note: If Command Register Bit 0 = 1 and a change of state on $\overline{DCD}$ occurs, $\overline{IRQ}$ will be set, and Status Register Bit 5 will reflect the new level. The state of $\overline{DCD}$ does not affect Transmitter operation, but must be low for the Receiver to operate.

Internal Organization

The Transmitter/Receiver sections of the SY6551 are depicted by the block diagram in Figure 5.

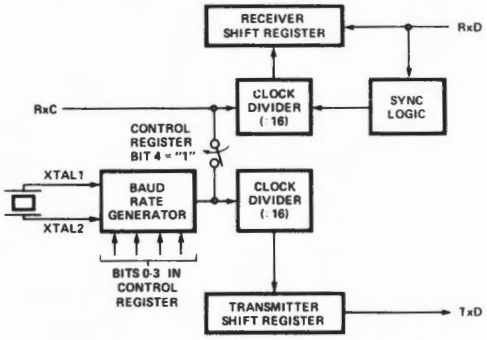


Figure 5. Transmitter/Receiver Clock Circuits

Bits 0-3 of the Control Register select the divisor used to generate the baud rate for the Transmitter. If the Receiver clock is to use the same baud rate as the Transmitter, then RxC becomes an output pin and can be used to slave other circuits to the SY6551.

Fig. 16.1b ACIA/Modem interface signals for 6551 ACIA.

Control Register

The Control Register is used to select the desired mode for the SY6551. The word length, number of stop bits, and clock controls are all determined by the Control Register, which is depicted in Figure 6.

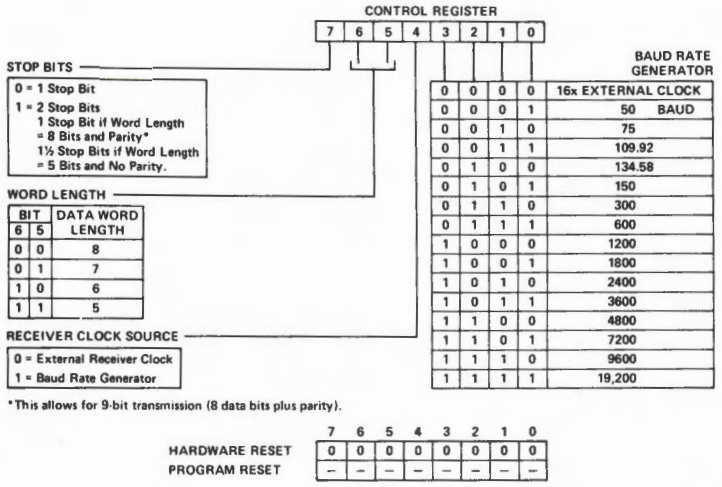


Fig. 16.1c 6551 ACIA Control Register. In the //c, it may be read from or written to at $C09B for the port 1 ACIA or at $C0AB for the port 2 ACIA.

Command Register

The Command Register is used to control Specific Transmit/Receive functions and is shown in Figure 7.

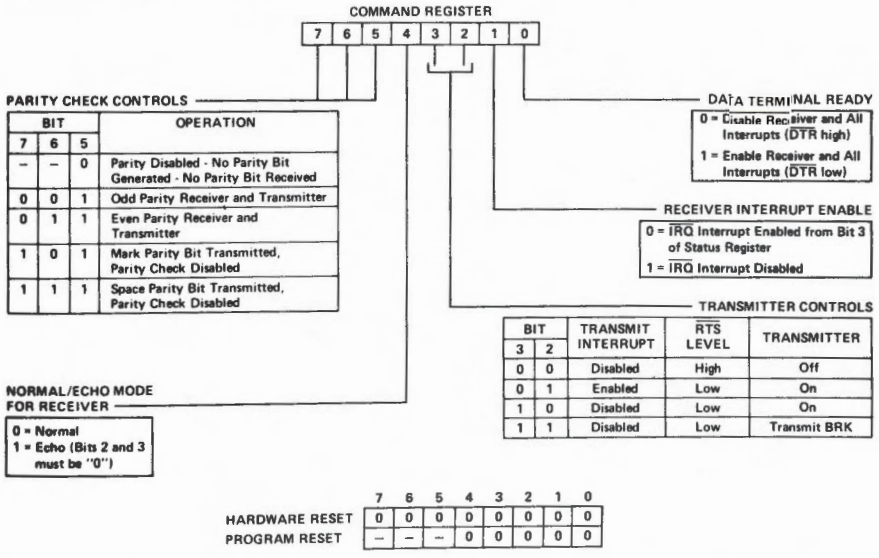
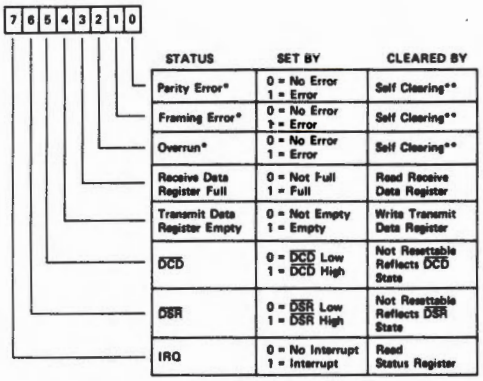


Fig. 16.1d 6551 ACIA Command Register. In the //c, it may be read from or written to at $C09A for the port 1 ACIA or at $C0AA for the port 2 ACIA.

Status Register

The Status Register is used to indicate to the processor the status of various SY6551 functions and is outlined in Figure 8.



STATUS	SET BY	CLEARED BY
Parity Error*	0 = No Error 1 = Error	Self Clearing**
Framing Error*	0 = No Error 1 = Error	Self Clearing**
Overrun*	0 = No Error 1 = Error	Self Clearing**
Receive Data Register Full	0 = Not Full 1 = Full	Read Receive Data Register
Transmit Data Register Empty	0 = Not Empty 1 = Empty	Write Transmit Data Register
$\overline{DCD}$	0 = $\overline{DCD}$ Low 1 = $\overline{DCD}$ High	Not Resettable Reflects $\overline{DCD}$ State
$\overline{DSR}$	0 = $\overline{DSR}$ Low 1 = $\overline{DSR}$ High	Not Resettable Reflects $\overline{DSR}$ State
IRQ	0 = No Interrupt 1 = Interrupt	Read Status Register

*NO INTERRUPT GENERATED FOR THESE CONDITIONS.
**CLEARED AUTOMATICALLY AFTER A READ OF RDR AND THE NEXT ERROR FREE RECEIPT OF DATA.

	7	6	5	4	3	2	1	0
HARDWARE RESET	0	–	–	1	0	0	0	0
PROGRAM RESET	–	–	–	–	–	0	–	–

Transmit and Receive Data Registers

These registers are used as temporary data storage for the 6551 Transmit and Receive circuits. The Transmit Data Register is characterized as follows:

- Bit 0 is the leading bit to be transmitted.
- Unused data bits are the high-order bits and are "don't care" for transmission.

The Receive Data Register is characterized in a similar fashion:

- Bit 0 is the leading bit received.
- Unused data bits are the high-order bits and are "0" for the receiver.
- Parity bits are not contained in the Receive Data Register, but are stripped-off after being used for external parity checking. Parity and all unused high-order bits are "0".

Figure 9 illustrates a single transmitted or received data word, for the example of 8 data bits, parity, and 1 stop bit.

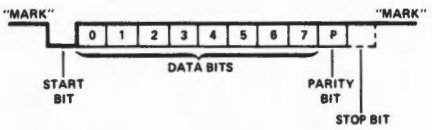


Fig. 16.1e 6551 ACIA Status Register. In the //c, it may be read from or written to at $C099 for the port 1 ACIA or $C0A9 for the port 2 ACIA.

Framing, Serial Format and Baud Rates

Once an ACIA has gotten hold of an ASCII code and done whatever it's supposed to do to the eighth bit, it has then got the problem of getting this set of eight bits onto a single wire for transmission on a phone line. To do this, an asynchronous communication system creates a little pattern called a "frame." In a typical setup, it turns on its output for 3.3 milliseconds, then turns it off for eight more intervals of 3.3 milliseconds each (off for $8 \times 3.3 = 26.6$ milliseconds), then turns it on again for a closing burst, also about 3.3 milliseconds in duration. The initial burst is called a "start bit," the eight "off" periods are called a "frame," and the closing burst is called a "stop bit" (see Figure 16.1e).

In the standard conventions of asynchronous communications, the frame is considered to have eight "cells" in it. These cells are not physically marked off from each other, but rather reflect the time allowed (3.3 milliseconds apiece). Each of the cells can be used to communicate one bit of data. The disk drive system also uses a scheme with cells in a serial stream, but on a floppy disk a marker is placed between each cell. There are no physical or electrical cell markers in asynchronous communications.

As long as both the sender and receiver are both using the same 3.3 millisecond intervals, the receiver is able to know when a character frame has arrived, and when each bit should appear within the frame. If the code that is sent within the frame is "00010001," there will be no ambiguity about when the data begins. The three leading zeros in this example will be recognized as zeros because nothing happens in the signal line during the first three cells of the frame.

To create a frame for transmission, one part of the ACIA outputs a start bit, and then a different part of the ACIA which is connected to the same line drops the data bits into the time cells, one after another. Within a given cell, it does nothing to the line if the data is zero, but puts a 3.3 millisecond pulse onto the line if the data bit is a one. Finally, the framing section of the ACIA outputs a stop bit, and the device prepares to send the next character.

Each character therefore requires 10 intervals, one start bit, eight bits for the parity + ASCII code, and one stop bit. If each bit is allotted 3.3 milliseconds, and these 10 bit character codes are sent one after another in rapid succession, then there will be time for 300 bits in each second. This is called a "300 baud" transmission rate. But, as you can see, the critical timing element is the duration of the framing and data cells. An instruction to the ACIA that it should operate at 300 baud (see Figure 16.1c) simply informs it that it should use 3.3 millisecond cells even if it has only one character to send every 10 minutes.

RS-232C Data Signals

The 10 bit character code that emerges from the ACIA is a fairly standard computer signal in that it is either 0 volts or 5 volts, and it is compatible with standard "TTL" chips (see Chapter 13. However, although TTL signals are just fine running from chip to chip inside a computer, they tend to get overwhelmed if they travel more than two or three feet in a cable outside the computer.

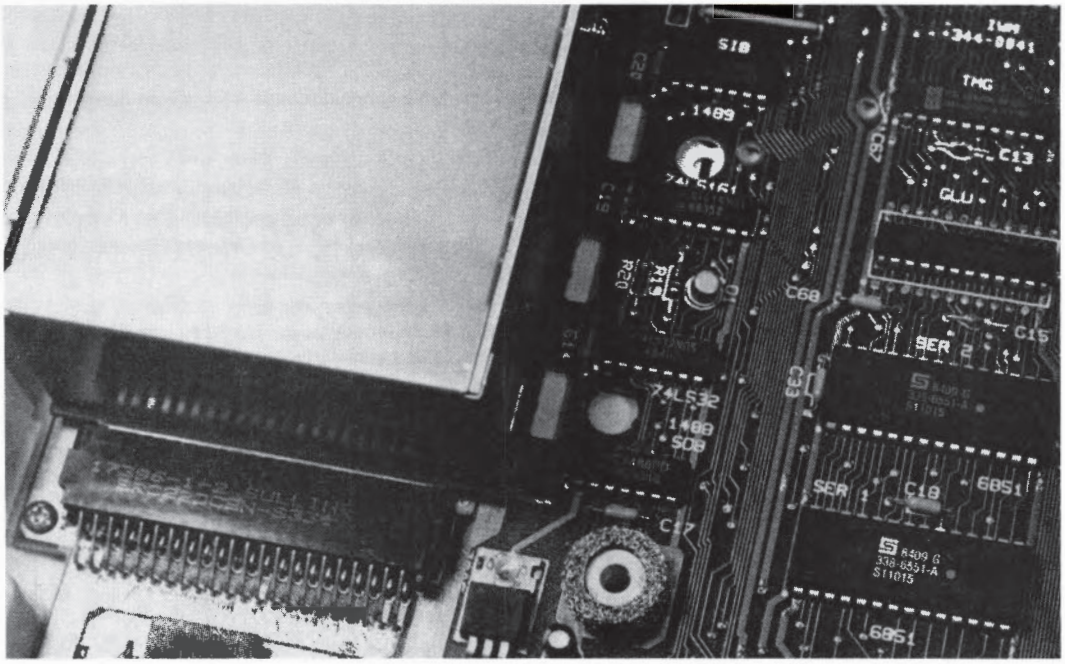


Fig. 16.2 The //c serial interface hardware includes two 6551 ACIA's, an MC 1488 and MC 1489 to handle the RS-232 electrical interface and a 74LS161 chip to provide the baud clock rate which it gets by dividing down the 14 MHz system clock frequently.

The RS-232C system was intended to securely carry asynchronous serial codes over distances of up to about 50 feet. And although the RS-232C standard was designed long before the birth of TTL chips, it serves nicely for handling the external tasks of what is internally a TTL system. A chip called the Motorola 1488 is used in the AIO-II, the Apple Super Serial Card and in the //c (see Figure 16.2) to do the conversion from TTL signals to RS-232 signals. Although RS-232C allows considerable leeway, most computer manufacturers use a standard signal system in which any pulse which is +5 volts in the computer is sent out as -12 volts, and any signal which is 0 volts in the computer, comes out of the 1488 at +12 volts.

As a voltage pulse travels along a wire, it encounters some fixed amount of resistance per foot of wire and this causes the voltage to drop steadily. Once a TTL signal has dropped from +5 volts to about +2.5 volts, it gets fairly unreadable. In RS-232C, however, the starting signals range from +12 volts to -12 volts and the typical RS-232C receiver is designed to be able to distinguish +5 from -5 volts, thus allowing for a loss of seven volts in either line. In a TTL line, a one volt increase of the 0 volt signal together with a 1 volt decrease of the 5 volt signal can be fatal to transmission. RS-232C transmissions are much more rugged.

Once an RS-232C signal arrives and is detected as positive or negative, a Motorola 1489 chip can convert back into standard TTL levels for the receiving ACIA.

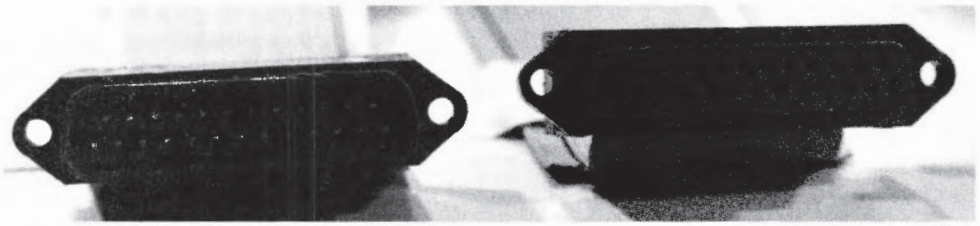


Fig. 16.3a DB-25 connectors for RS-232C communications. Male connector on left, female on right.

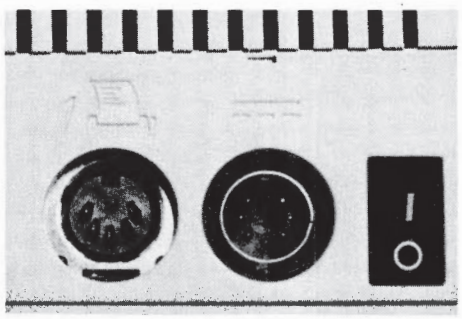
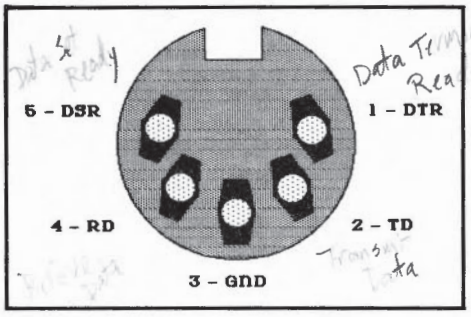


Fig. 16.3b A 5-pin DIN connector used for the //c RS-232C serial ports.

Fig. 16.3c Pinout for the RS-232C signals on the //c DIN connector.



The RS-232C Mechanical Connector Problem

In addition to agreeing on the permissible voltage levels for the wires which carry the data, the RS-232C committee also agreed to a standard for the shape and number of pins for the plugs and receptacles. The type of connector is called a "DB-25" or "25 pin subminiature D connector" (see Figure 16.3a).

The "male" version of the connector has the pins and the "female" version has the little holes. It was recommended that a female connector be placed on the modem and a male connector be used on the computer; however, pins projecting from an unplugged connector tend to get bent or short circuited, so many computer manufacturers have put female connectors on the computer end as well. This is no problem if you connect the two with a cable that has a male connector on both of its ends. Apple supplies the Super Serial Card with a female connector, SSM gives you a choice with the AIO-II, and if all else fails you can always run down to Radio Shack to get the appropriate parts to make your own.

The //c serial ports both have five pin DIN connectors (see Figure 16.3b). You'll do best to buy cables specially made for the //c because there are several different kinds of five pin DIN connectors and no standard for what wire goes on which pin. Ideally, you want a cable with a male DIN connector on one end and a male DB-25 connector on the other end.

The RS-232C Signal Connector Problem

The real "connector problem" with RS-232C has nothing to do with male versus female, however, but is a little bit more subtle and a lot harder to detect. The RS-232C committee assumed their standard would be used primarily for connecting terminals to modems and for connecting computers to modems. Terminals and computers were called Data Terminal Equipment (DTE), and modems were called Data Communications Equipment (DCE). All of the signal wires used in the RS-232C standard are allowed to carry information in one direction only. Some carry information from the DCE (modem) to the DTE (computer) and some go the other way.

Of the 25 pins available on the DB-25 connector, the primary concern for many applications is limited only to pins number 2 and number 3. Number 2 is intended to carry data signals from a DTE (computer) to a DCE (modem) and is called Transmit Data (TD). Pin 3 is for carry data in from the DCE (modem) to the DTE (computer) and is called Received Data or (RD). This is all simple and straightforward if you are connecting an Apple to a modem.

However, some Apple owners connect separate CRT terminals to their computers in order to take advantage of the detachable keyboard, and others wish to connect their machines directly to another computer without an intervening modem. In each of these situations, all parties involved will consider themselves to be DTEs. All will send out data on pin 2, yet no data will ever turn up on anyone's pin 3.

Much worse, most printers consider themselves to be terminals. Both the Apple //c and the Imagewriter are configured as terminals (DTE) and neither is a DCE. Once again, the best thing is to buy specially made cables which handle the problem for you. But note that you can't use the same //c cable for a modem and for a printer.

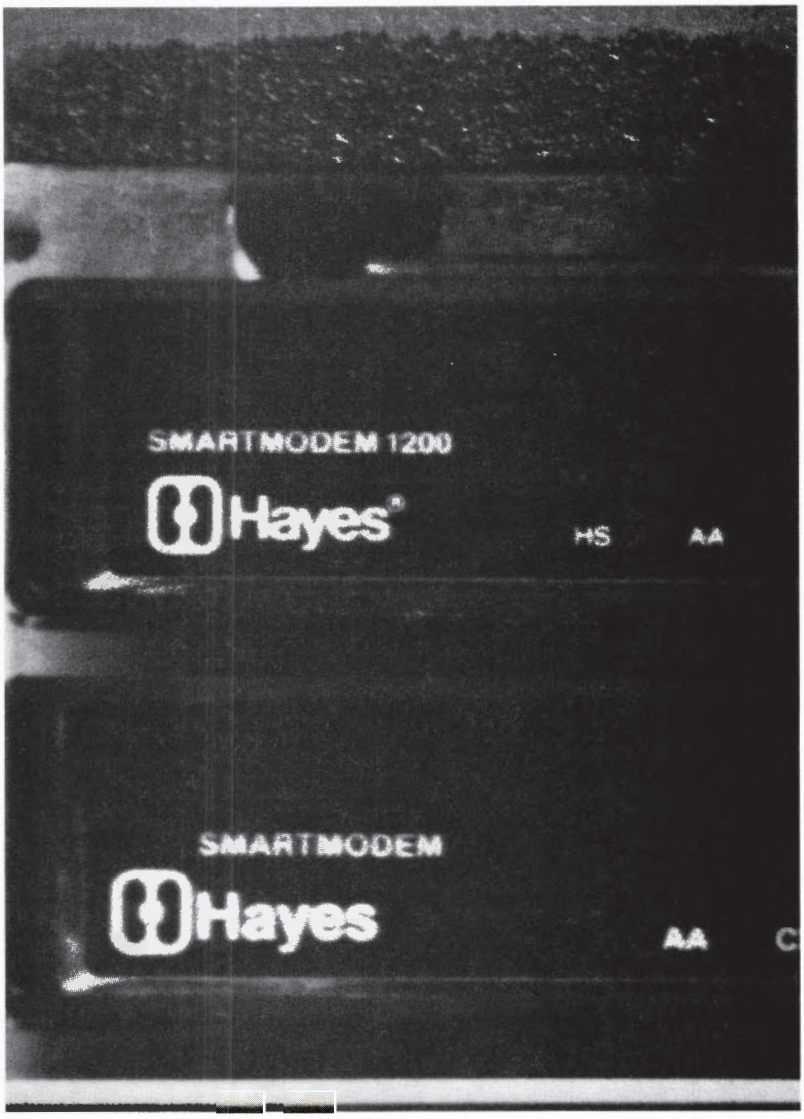
Handling the "Null Modem" Problem with Interface Cards

If you bought the old Apple serial card, it was set up as if the Apple were a DCE or modem. To connect to a real modem, you had to solder together a device called a "null modem" or "modem eliminator" which, among other things, connected line 3 (the TD for a DCE) from the Apple over onto line 2 (the TD for a DTE). The reason Apple chose this configuration is

because they expected most printers, mainframe computers and terminals to want to play the part of the DTE; so the Apple would have to pretend it was a modem (DCE). Many serial cards are still sold in this configuration, so M & R makes an item called an Adaptabox which simplifies the construction of a null modem.

The Super Serial Card from Apple has a little unit called a "jumper block" which you unplug, turn around, and replug to do the DCE to DTC conversion. The AIO-II from SSM has two different connectors on the card, one labled "modem" (for connecting to a modem) and one labeled "terminal" (for connecting to a printer, terminal or other computer). The only limitation on the AIO-II is that you can't use both at the same time, but this restriction is eased a bit because the AIO-II has a separate "Parallel" printer port which can be used to simultaneously print the data being passed through your modem.

Another company, California Computer Systems (CCS), sells two different serial cards; the 7710A, which is set up as a DCE, and the 7710D, which is set up as a DTE. Yet another variant is the CPS Multifunction card from Mountain Computer which can switch from DCE to DTE under software control, retaining the configuration for up to two years (battery backup memory) or until instructed to change again.


SMARTMODEM 1200
Hayes
HS
AA
SMARTMODEM
Hayes
AA

Chapter 17

Modems

Why Computers Need Modems

Until all the copper wire in the telephone system gets replaced with optical glass fiber, computer users are going to have to continue to endure major dependence on a connection and signal system which is grossly unsuited to digital data transfer. In networks (see Chapter 20), computers are able to exchange data at rates of up to four million baud. The telephone system as it is now arranged cannot tolerate signal changes faster than 600 baud. Yes, that was four million versus 600.

It's not that the telephone company has anything against computers. The very reason that computer users need the phone system is the same reason that it cannot adapt to their needs. Telephone wires go absolutely everywhere. We're talking about hundreds of millions of miles of copper, well into the gobs and gobs range. Whenever the phone company makes a change or improvement in its system, it has to see that the improvement will not either cut off vast numbers of phone users or force them to replace all the wires. Actually, things have gotten so intolerable that the replacement process has begun, in favor of lasers and fiber optics, but this is going to take decades.

The initial premise of the telephone system was that everything should be engineered for clean, crisp transmission of voice. No easy task, yet certainly well met. Voice is transmitted as an electronic image of oscillating sound waves. The majority of the important frequencies in human voice are in the 700 to 3000 Hz range that the phone system is designed to accept. When you have predictable kinds of continuous oscillations you can apply all kinds of analog magic to, for instance, interweave dozens of simultaneous phone conversations in a single wire and then sort them out at distribution points so the parties speaking never know they have anything other than their very own individual line.

This a wonderful trick, and we all appreciate it—most of the time. However, if you try to send digital on/off signals through this phone system you encounter gross failure. The phone system no longer seems so wonderful.

First, the wires have a small but finite capacitance (see Chapter 13), which means that in the first few instants of a voltage transition, current rushes to fill these effective capacitive plates. However, as the signal transitions become faster and faster, the fixed filling time becomes a greater and greater percentage of the total length of the signal burst. Above a certain frequency the signal spends all its time emptying and filling the capacitance of the wire. The sum effect of this is that above standard telephone frequencies, a signal just disappears into the wire near the transmission point and never makes it to the other end.

Going slow still doesn't solve the problem. Much of the wizardry of telephone switching, compressing and line sharing is quite toxic to discrete digital pulses and vice versa. Digital is not allowed and it won't work. If computers insist on conversing on the phone, they must generate signals which behave like voice but are still easily interpreted by machine. Enter the modem.

Modem Features

The simplest modems (i.e., Signalman, $99) do little more than sit between your serial card and your telephone receiver, patiently and slowly moving digital data onto and off of the phone lines. The most elaborate modems (i.e., Novation 212, Apple-CAT II, $800+) will, unattended, wake themselves up in the middle of the night, call another modem, transfer large files at high speed and with great accuracy, turn out the lights in your living room, answer incoming calls, collect any incoming modem messages for you to see later, and periodically do complete checks on the integrity of their internal circuitry.

If you've never owned a modem, you may be amazed at the thought of spending over $500 for a phone hook-up. However, for most business and academic telecommunication needs, a very good case can be made for buying a full featured modem such as the Hayes Smartmodem 1200 ($700), and home users should give full consideration to their potential needs before buying a minimal modem.

Modulation, Demodulation and Carriers

The purpose of a modem (which is a contraction of the term modulator/demodulator) is to stand between the digital world of the computer and the voice world of telephone and mediate between them. The simpler modems which transmit information at 110 or 300 baud use a very simple conversion system, generating two different tones to represent the digital on or off. At higher speeds of 1200 bits per second, modems are forced to engage in much more elegant magic of their own to slip high speed information through a low speed conduction system.

Low Speed Modulation

The low speed (up to 300 baud) system actually dates back to early days of sending Morse code over telegraph lines. The two tones are just oscillations at two different frequencies and so the system is called Frequency Shift Keying (FSK) with the "key" referring to the little lever you tap on to generate Morse code. As the bits arrive at a modem from an RS-232C TD line, the modem responds by generating one tone for zeros and a different tone for ones.

RS-232C signals are digital, and one consequence of this is that one wire cannot carry information in two directions at exactly the same time. If you want to communicate in two directions in one digital wire, you have to time everything very carefully so that the two sources are unambiguously on at different times. This is all very demanding and RS-232C avoids the problem by using two different wires, one (wire 2, TD) from the DTE (computer) to the DCE (modem) and one from the DCE to the DTE (wire 3, RD).

In a telephone line, however, no such restrictions apply. This is because telephone information signals are similar to sound and voice. When you are in a room with several people talking

at the same time, you may not be able to pay attention to all of them, but your ears have no trouble taking what is really a single complex sound wave and decomposing it into several distinct and identifiable voices. As is well known, telephone conversation can take place in two directions simultaneously, but it should be clear that both speakers' voice signals are completely intermingled in a complex pattern on a single wire.

Similarly, modems are able to converse in two directions on the same wire as long as the two have different sounding voices, so to speak. The modem which originates the conversation is required to "sing" its tones with a low pitched voice, while the answering modem sings with a high pitched voice. When your modem contacts another modem, it maintains a low voice through out the conversation. When another modem calls you, then your modem switches to its high pitched voice. If you've ever listened to a calling modem, you've no doubt heard the steady lower tone called the "originate carrier" tone.

In the United States there is a standard called Bell 103/113 which specifies what tones should be used. The agreed on frequency for the originate carrier in the U.S. is 1070 Hz, and the agreed on frequency for the answer carrier tone is 2025 Hz. These are the two voices. When the two modems are both "on line," both signals exist in the same telephone wire, but, like your ear, the listening circuits of the modem have no problem telling the two apart.

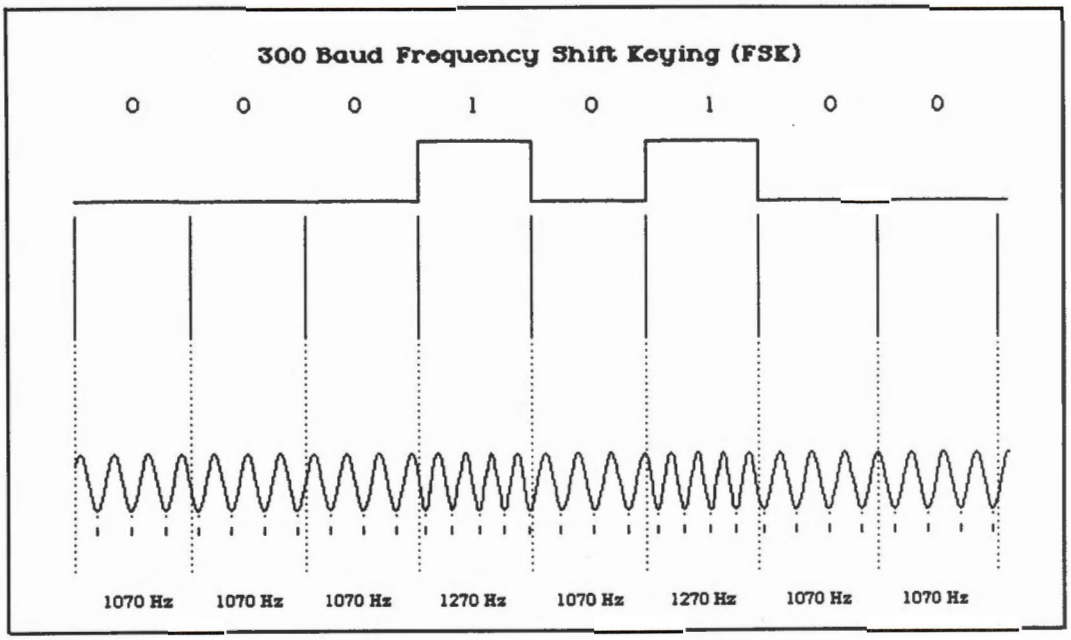


Fig. 17.1 FSK modulation.

Whenever the originating modem has a one to transmit, it shifts frequencies a little to 1270 Hz, a zero brings it abruptly back down to the deeper tone at 1070 Hz (see Figure 17.1). The originating modem shifts between a zero or "space" state of 2025 Hz and a one or "mark" state of 2225 Hz. Two way communication in the same line are referred to as "Full Duplex," as opposed to "Half Duplex" communication where only one kind of tone is used and communication proceeds in one direction only.

International communications are conducted at a slightly different frequency using a standard called "V.21" established by the Comite Consultatif International Telephonique et Telegraphique (CCITT). Why must the French be different? Well, there are actually supposed to be accents over several of the letters in the name of that international organization, but there is no ASCII representation for accented letters. Our American codes leave much of the world out in the cold, even European countries which use an alphabet very much like our own. American standards usually aren't sufficient, so the CCITT goes about its business being concerned with the special needs of a much broader community.

Acoustic Coupling

The simplest and oldest way of getting the modem tones onto the telephone line is to actually generate the tones as sounds and then press a regular telephone receiver up against the modem's speaker. Back in the days before modular phone jacks, you had to cut and splice wires if you wanted a direct electrical connection between your modem and the phone line. This was not popular or convenient, so "acoustic" modems which actually relied on audible tones were very popular.

These devices are still useful if you need to quickly connect a modem in a place without both modular jacks and some sort of dual connection. Typically, you need to dial a number to get a carrier tone, and then connect your modem instead of your telephone. This is a simple trick if you can just plop the receiver down in the cradle of an acoustic modem. For this reason, several manufacturers of portable computers have surrendered precious weight and space to include an acoustic modem.

Acoustic modems which operate with the Bell 103/113 standard are available from Novation and from Multi-Tech. The big problem with modems based on acoustic couplers is that they have comparatively poor performance at secure data transmission. Someone talking in the room might generate confusing tones. Novation sells a device called the Super Mike which replaces and improves upon the microphone in your telephone handset, but if at all possible, acoustic couplers should be avoided.

300 Baud versus 1200 Bps

The major choice for most folks buying modems is between the slow 300 baud (30 characters per second) or the faster 1200 bps (120 characters per second) machines. In fact, there is little good justification for choosing a 300 baud modem unless you expect to use it only a few times a year for fairly light work. A 300 baud modem costs less the day you buy it, but when you are using a modem, you pay for telephone use by the minute and when you are contacting a database such as The Source, you pay by the minute once again.

Everything takes four times longer with a 300 baud modem, so not only does the screen fill with painful slowness, and your telephone line stay busy for extra hours, but all of your time charges from the phone company, and some of them from the places you call, will be four times greater. This will cause you to cut back on your use of telecommunications and that certainly defeats your original purpose. The frustratingly slow response of your computer, the endless busy signals for people trying to reach you, and the high time charges together conspire to cause many owners of 300 baud devices to put aside their first modem and buy a second one which runs at 1200 bps. Only Novation provides a means of upgrading a 300 baud modem to 1200 bps (Apple-CAT II with 212 upgrade).

In the past, many Apple owners purchased 300 baud modems simply because none of the easily installed internal modem cards could provide 1200 bps communications. The Era 2 1200 bps internal modem card from Microcom, which is now available, should convince many Apple II, II+ and //e owners to buy 1200 bps. The Era 2 is comparatively inexpensive for a 1200 bps system, and since you don't need an interface card, it is far and away the lowest cost path to high speed communication. It includes terminal emulation software and special data protocols for communications with computers from other manufacturers.

High Speed Modulation

The FSK system used for 300 baud modems is useless for transmission at 1200 bps. Since telephone communications permit signals from 700 Hz to 3000 Hz, it will no doubt be impossible to understand why FSK modulation at 1200 bps won't work.

The top diagram in Figure 17.2 is a representation of the range of frequencies accepted by the phone line. It is divided down the center at 1850 Hz, with all the lower frequencies being available for the originating modem and all the higher frequencies being reserved for the answering modem. The originate carrier tone of 1070 Hz sits nearly in the middle of the lower half, while the answer carrier tone of 2025 is near the middle of the upper half. As you can see, the small shift between 1070 and 1270 Hz leaves most of the region between the bottom (700 Hz) and the top of the originate zone (1850 Hz) untouched and presumably quiet, and the situation in the answer zone is similar.

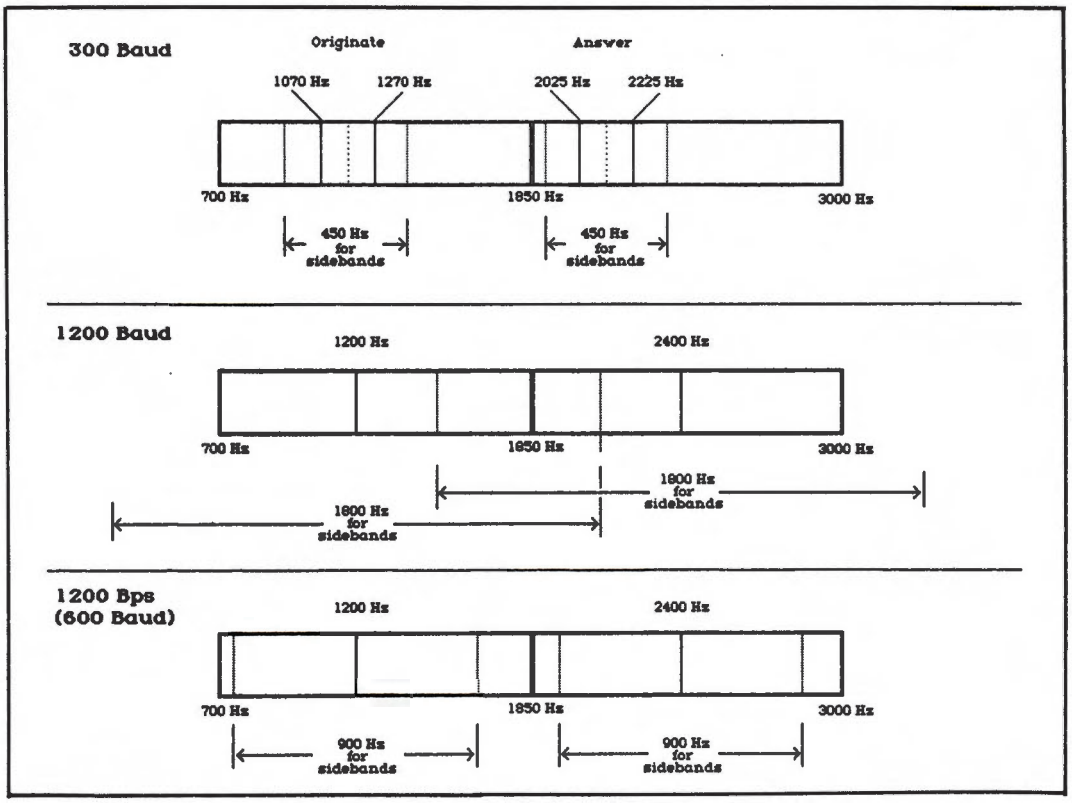


Fig. 17.2 Telephone lines permit frequencies between 700 and 3000 Hz. When a tone is modulated, it generates "sidebands" which require space in the frequency spectrum.

In reality, however, the act of modulating the frequency of a tone causes a few ringing frequencies to appear. These are a kind of harmonic, much as in music. They are called "sidebands," and they are extremely important in radio and TV broadcasting. It is sufficient to appreciate that they cause disturbances at a variety of frequencies around the central carrier tone. The further from the carrier, the lower the amplitude of the disturbance. In practice, if the modulation is going on at 300 Hz then the sidebands will cause substantial disturbances over a range of 1.5 times 300 equals 450 Hz. You must therefore set aside a band of frequencies, 450 Hz in width, around the center of the mark and space frequencies. Figure 17.2 also shows the true space required by the originate and answer signals, with sideband disturbances included.

Now that you know about sidebands, consider what would happen if you tried to modulate at 1200 Hz. As is shown in Figure 17.2, there would be overlap and interference between the two "voices." In actual practice, FSK can be done at 1200 Hz if the line is half duplex, that is, if communication is only tolerated in one direction. For the most part, however, FSK is not useful. The maximum reasonable modulation frequency is 600 Hz. This is what is meant when it is said that a telephone line can only be modulated at 600 baud.

Phase Shift Keying for High Speed Communication

Modems that can handle two way communication at 1200 bps are much more expensive than 300 baud modems. The extra cost is not just to pay for the same circuit running a little faster. Rather, a 1200 bps modem requires a much more sophisticated means of modulating data, and the electronics involved is far more elaborate. Good quality stand alone modems for 300 baud communications typically cost about $200, while 1200 bps modems of similar external appearance list for $600.

The basic solution to putting 1200 bps communications into a 600 baud phone line is obvious, just put two bits in every baud. What this might mean is that you use four different tones instead of just two tones. You would not modulate any faster, but each signal would convey a richer amount of information. There are four possible pairs of bits; 00, 01, 10, and 11 (four "dibits"). If each one of the four is assigned a different signal then you could use any possible combination of ten bits as a series of five dibit signals.

In actual practice, it is not possible to use FSK with four different frequencies. As you can see from Figure 17.2, 600 baud is already calling it close on sidebands, and using four different frequencies makes the sidebands too wide. Therefore, communications at 1200 bps do not modulate frequency. The originator sends a carrier at 1200 Hz, and the answering modem uses 2400 Hz, but the frequencies are not varied.

What is modulated for 1200 bps communications is the phase of the signal (see discussion of phase in Chapter 7). If the dibit is 00, the carrier is shifted to begin each signal 1/4 a cycle late (+90 degrees), 01 is set to no phase shift, 10 starts a half cycle late (+180 degrees), and 11 starts 1/4 cycle early (-90 degrees). This system is called Phase Shift Keying (PSK). In this way, modulation is performed at 600 baud, each baud describes two bits, and everything is done at a single frequency.

To demodulate a PSK signal is a bit tricky since you must know the true unshifted phase with which to compare each dibit. This calls for one last trick, which is to sneak in a reference signal for timing the phase shifts. To do this, a 1200 bps modem collects in asynchronous data and attempts to send it as long streams in continuous synchrony. If new characters come to the modem from the computer too slowly or too fast, it is able to remove or add a few extra stop bits to keep its pulsed output steady and regular. Finally, the data bits are shuffled around

amongst each other in a way which is both sufficiently predictable to permit reconstruction by the receiving modem, yet also sufficiently irregular that they won't confuse the interpretation of the timing signal.

This is truly arcane stuff, but it is sufficient to note that 1200 bps transmission is a very different beast than 300 baud transmission. Further, there are even more complex modulation schemes which are so bizarre that few engineers can understand them well, but which have the balancing merit of permitting transmission at speeds up to 9600 bps without the telephone wires ever knowing what is zipping by within them.

Bell 212 versus Bell 202

Buyers should also be aware of the fact that there are two very different kinds of 1200 bps modems on the market. As explained earlier, you cannot conduct "full duplex" (two way) communications on the phone line at 1200 bps unless you do some very fancy electronic tricks. There is a standard for the trickery which is called the "Bell 212" system. However, if you use only half duplex (one way) conversation, then transmission can be achieved by the same simple modulation technique used for 300 baud communications.

These half duplex 1200 baud systems are called "Bell 202" compatible. The electronics in the modem is much simpler, so these 1200 baud modems are much less expensive. However, they constitute one large monkey wrench in that they are not the standard—you may not be able to use them for most communications needs. The Sup'R Access-1 from M & R uses Bell 202 (note: this device from M & R offers several other more useful features), and it crops up here and there among other equipment; so be careful when someone offers a real cheap "1200 baud" modem.

1200 bps Problems for the //e and //c

Quite inpependent of the details of the modem itself, some Apple owners may have difficulties using 1200 bps modems because of particular quirks of some models of the Apple. One sort of problem applies to some Apple //e's and the other applies to some Apple //c's.

Slow Video Echo on the //e

The problem in the //e has to do with the 80 column video system. In most uses of a modem, each character that arrives from the telephone line is "echoed" for display on the video screen. As characters come pouring in, the //e does just fine for a while, steadily filling the screen with incoming text. However, once the screen has 24 full lines of text on it, the Apple has to "scroll" the display. In the //e 80 column sytem, scrolling up by one line keeps the 6502 busy long enough so that it will miss one or two incoming characters before it gets done with the screen. With the screen full, it has to scroll after each line, and the situation becomes intolerable.

If you use a completely different kind of 80 column card, of the sort that Apple II and II+ owners put in slot 3, then you will have no problem. Those cards have a specialized CRT controller chips which scrolls almost instantaneously. However, if you still want to use extra memory in the auxiliary slot, you can only use the Ultraterm card from Videx, since other, less expensive cards, require you to remove all cards from the auxiliary slot.

Another way to solve the problem is to use a serial interface card that generates "interrupts" to stop the scrolling process for an instant whenever a new character needs to be captured in from the modem. This requires specially designed communications software, and won't work on all //e's. Older //e's (bought in 1983) automatically disable the interrupt system whenever

scrolling begins. //e's from the first half of 1984 permit interrupts, but only with ProDOS software. Finally, beginning in the summer of 1984, a third version of the Monitor and I/O ROM chips was released that fully integrated interrupts. This version of the ROMs can be installed in older //e's as an upgrade.

The least expensive option is to choose software that has specially built routines that either have their own fast scrolling routines or capture characters without always sending them immediately for display. Ask the company that sells the software before you buy. If they don't know what you're talking about, their software probably won't work for your machine.

However, rather than buying a new 80 column card or new software, consider the following. The Practical Peripherals external serial printer buffer, the MBIS (see Chapter 18, Figure 18.12a) is bidirectional, so incoming files from the modem can be captured by the buffer while you attend to other business, and subsequently brought into the //e at a speed it can handle. When you're shipping out large files there may be no need to view the text on the screen, so you can send it to the buffer at very high speed, then dial up, log on, and transmit at a moderately fast 1200 bps from the buffer. When you're done, just disconnect the modem and reconnect your printer.

The //c video system is very similar to the //e video system, but the scrolling routines have been rewritten in 65C02 machine language, which is just enough faster than 6502 code that the whole thing works fine. Better yet, the //c has an elegant and elaborate interrupt and buffer system (see Chapter 18) which is activated automatically when the modem port is used in "terminal" configuration. This interrupt system comes built-in to the //c's hardware and firmware routines, so your average communications software package has to work very hard to override it and muck things up.

Inaccurate Baud Clock on the //c

The problem in early //c's is that when you configure the modem port to run at 1200 bps, it actually runs at about 1165 bps (three percent too slow). What went wrong? Well, the //c uses a Synertek 6551 ACIA (Asynchronous Commmunication Interface Adapter) to manage its serial ports; one for the printer port and one for the modem port. One of the features of the 6551 is that if you provide it with a single clock input at a frequency of 1.8432 MHz, then it will generate just about any baud rate you want (see Chapter 16, Figure 17.1c).

To get that input frequency, you could put a special crystal on the motherboard to generate it. However, Apple engineers chose to minimize the risk of electrical interference and to reduce the number of parts in the //c. Instead of a crystal, the //c has a 74LS 161 "four bit counter" which starts with the Apple master clock frequency of 14.31818 MHz, and effectively divides it by eight to get an output frequency of 1.7898 MHz. This is very close to what the 6551 needs. Close enough for 300 baud communication and almost close enough for 1200 bps communication.

The engineers tested this with 1200 bps modems and it worked perfectly well. Thus the design went forward and production began. Later, they discovered that if your communications protocol uses eight bit data, then framing errors can occur (see Chapter 16).

What this means is that with each bit in the character frame, the 6551 gets a little bit further out of synchrony. Even the seventh bit makes it through OK, but by the time the eighth bit arrives, things are too far out of whack. Most communications use seven bit data, so you'll be OK. However, if you are having mysterious problems with data transmission errors at 1200 bps, you may want to talk to your dealer about a trade or upgrade to a later //c which has the 1.8432 MHz crystal.

Three Kinds of Modem Control Systems

In addition to your 300 versus 1200 choice, you need to select from among a very wide range of devices, all referred to as "modems." All of them do modulation and demodulation, but most of them do quite a bit more.

In this case, most of the differences have to do with the quality of interaction between the computer and the modem. This refers to such features as automatic dialing, detecting busy signals and passing the bad news back to the computer, etc., rather than to what happens to the actual data to be launched into the telephone line.

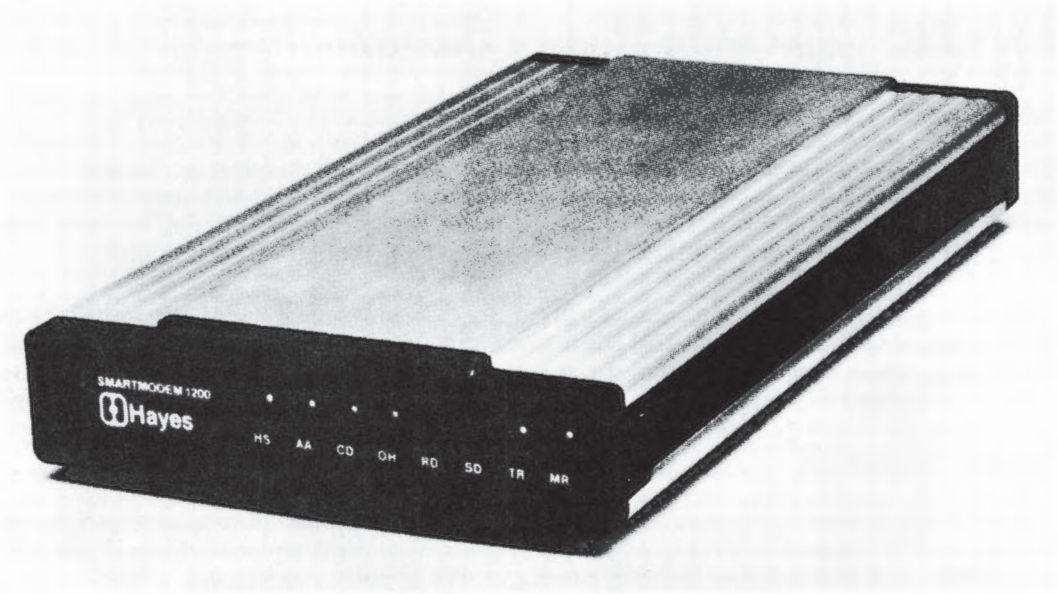


Fig. 17.3 Hayes Smartmodem 1200.

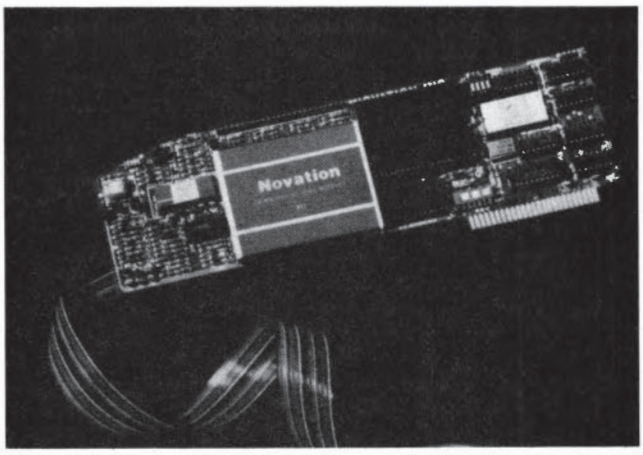


Fig. 17.4 Apple-Cat II 300 baud modem card.

From this point of view, there are three different kinds of modems. The first kind relies entirely on the control lines provided by the RS-232C system. The second kind includes the various external "smart" modems (see Figure 17.3) that contain their own microprocessor and can exchange detailed commands and information reports.

The third kind is available only for computers with slots (such as the Apple II, II+ and //e) and involves complete and direct control of the modem system by the microprocessor. These internal "modem cards" (see Figure 17.4) usually have features quite similar to those of a smart modem except that the Apple's 6502 controls the features directly rather than relying on the modem's own microprocessor.

RS-232C Signals for Modem Control

You will recall from Chapter 16 that pin 2 (Transmit Data) and pin 3 (Receive Data) have the fundamental roles of passing information back and forth between the ACIA (which manages the coding operation) and the modulator/demodulator system. All modems use RD and TD; in fact, if you look closely at a Hayes Micromodem II card, you will see the same 6850 ACIA that is used on the AIO-II serial card (although on the Hayes card the RD and TD lines only travel for about a quarter of an inch, and they are never actually in RS-232C form).

Most external modems also use a second pair of RS-232C pins, one is called Request To Send (RTS; pin 4) and the second is called Clear To Send (CTS; pin 5; see Figure 17.5). The RTS line is an agreed upon single purpose control line through which the computer tells the modem to turn on its outgoing carrier tone. The modem always responds by turning on the signal, and once it has it going steady, it reports a confirmation back to the computer by sending a signal on the CTS line.

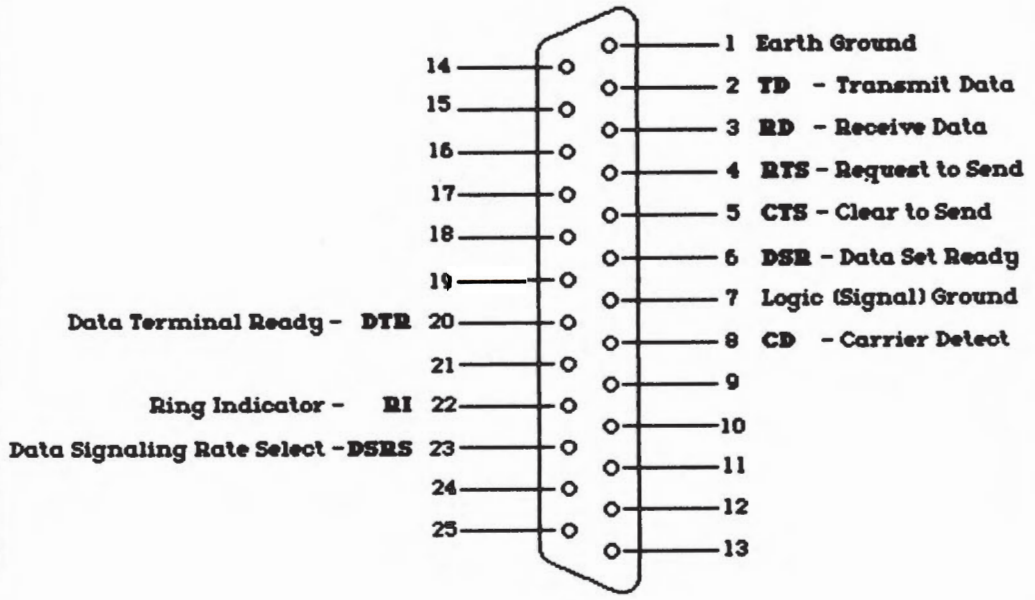
This RTS/CTS system is the computer's way of turning on the modem. A computer may decide to send an RTS command for one of two reasons: it wants to make a phone call, or it suspects some other computer is trying to get in touch with it and wishes to respond.

There is a third and crucial signal line in the RS-232C connector which turns RTS/CTS into a viable and functional command system. This third command line is called Carrier Detect (CD; pin 8—its full formal name is actually received line signal indicator, but CD or DCD are the common designations, and it's always on pin 8).

When a modem has got its power turned on, and it knows its computer is paying attention to it, it listens for incoming carrier tones on the phone line. If one comes in, it turns on the CD wire. This lets the computer know that some other computer is ready to exchange information.

Making a Connection with a Simple (RTS/CTS) Modem

If your computer were trying to get in touch with some other computer, its first step would be to turn on RTS and wait for its own modem to respond with CTS. The signal would be sent out to the appropriate destination computer. The destination computer would first find out about all this when it noticed that the CD line from its modem was on. If the destination computer has nothing better to do, it would turn on its own RTS. Its modem would send a CTS back to it and cause an answering carrier to be sent back to the first computer. At this point, the originating modem would hear the responding carrier and send a CD signal to its own computer.



Other Pin Assignments:

9 - Reserved
10 - Reserved
11 - Unassigned
12 - Secondary CD
13 - Secondary CTS
14 - Secondary TD
15 - Transmit Clock (to DTE)
16 - Secondary Receive Data
17 - Receive Clock
18 - Unassigned
19 - Secondary RTS
21 - Signal Quality Detect
24 - Transmit Clock (to DCE)
25 - Unassigned

Fig. 17.5 RS-232C signal assignments on DB-25 connector.

At the end of all this, both of the computer/modem pairs would have all of RTS, CTS and CD turned on. Whenever this situation occurs, communication can begin via RD, TD and the modulator/demodulator at each end of the line.

Acoustic Modems

This sort of RTS/CTS and CD system is built into virtually all modems and is a part of the 6850 ACIA. These signals are used in acoustic modems such as the Signalman (from Anchor), the CAT (from Novation), and the FM 30 (from Multi-Tech). In acoustic modems, the tone from the modulator is actually generated as a sound from a speaker, and the microphone in the telephone handset converts it back into electrical signals.

You may well be thinking: "Wait a second, who dials the number, who answers the phone?" Well, with an CTS/RTS modem, the answer is "you." When you're dealing with acoustic modems of this kind, it's no big deal to first get the RTS/CTS thing going so a whistle is coming out of the acoustic modem's microphone, and then to pick up any phone, dial the number, and then place the handset into the acoustic coupler's cradle. At the other end, the phone rings, some informed person picks it up, hears the carrier tone, gets their system to do the RTS/CTS thing, and then places their handset into their acoustic modem's cradle. This process is simplified a little bit if you are calling a big database—these companies usually just leave their modems on generating carrier tones.

Direct Connect Modems

As noted earlier, however, acoustic modems are subject to noise, interference and poor integrity of data transmission. Novation was one of the first companies to get approval from the Federal Communications Commission (FCC) to use a direct electrical connection between a modem and the phone wire. The use of "FCC part 68" for a direct connection brought about an enormous improvement in reliability of transmission, and made it possible to contemplate the construction of higher speed modems.

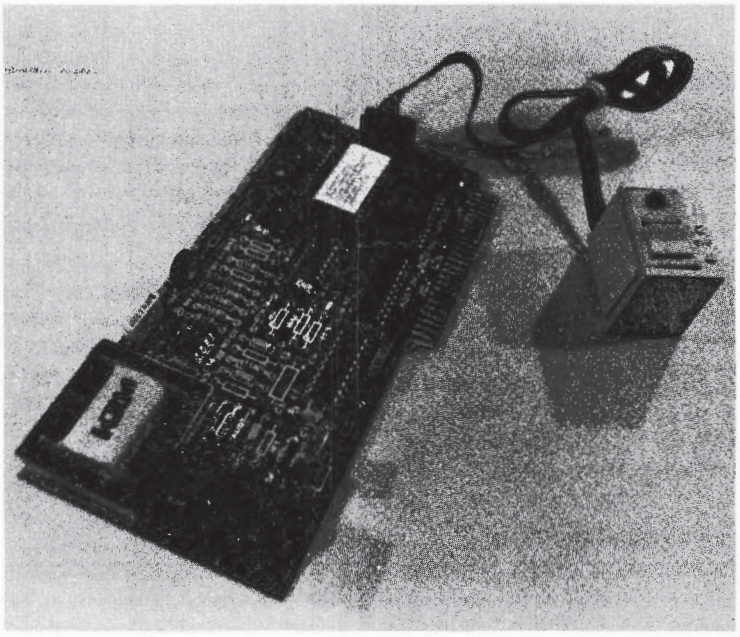


Fig. 17.6 Networker, a no frills, 300 baud modem.

There are a few comparatively inexpensive and reliable "direct connect" modems which operate with the RTS/CTS and CD control system, including the Novation D-CAT and the Sup'R Access-1 from M & R (which also offers switching among six different RS-232C ports). However, direct connect modems with this simple command system have never been very popular with microcomputer owners. The problem in part is that, compared to acoustic modems, they are a little more complicated to deal with when you are dialing up to make a connection. They require, minimally, a "modular jack" type of connection to the phone line and a little more attention to details. Nonetheless, if the cost of modems has been discouraging you from getting involved in communications, the bare bones, low cost ($130) direct connect Networker modem card (see Figure 17.6) from Zoom Telephonics might be worth looking into.

But the real reason these modems are not popular is that once a direct connection has been established, it enters the realm of possibility that the modem itself could do the dialing and the answering. Enter the "Auto-dial/Auto-answer direct connect modem."

Auto-Dial/Auto-Answer Modems

Nearly all direct connect modems have the ability of doing what amounts to taking the phone off the hook to answer it when its starts to ring, as well as the ability to make a phone call by electronically "taking the phone off the hook" and then electronically "dialing a number." The thorny problem is the "auto-answer" mode.

The ideal use of an auto-answer modem is to just leave it connected to your phone line so that it will answer whenever a modem calls. The problem is that you don't know in advance if it's a human or a modem that's calling and you may not want all your incoming calls answered with a high pitched squeal. The RS-232C convention provides three signal lines intended to deal with this sort of problem. However, although there are several "auto-answer/auto-dial" modems on the market that rely solely on the RS-232C conventions, these are not really sufficient for a smoothly functioning system.

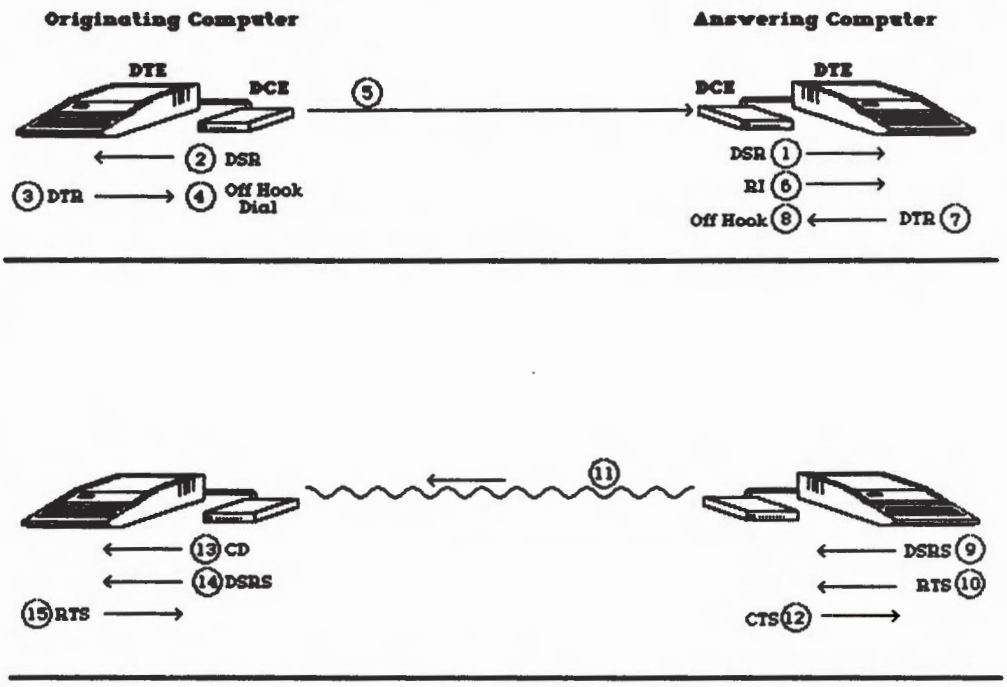
An auto-dial/auto-answer modem is typically (but not always) provided with a switch that lets you disable the modulator/demodulator but leaves the on hook/off hook system, as well as the dialing system, working. This feature lets you select a phone number from a list in your computer's memory by just typing in the name of the person you would like to speak to. Your software looks up the number and causes the modem to dial it for you automatically. Because the modulator/demodulator is turned off, you can pick up the handset on a telephone connected to the same line and just begin talking without any annoying carrier tone squeal.

When a modem is placed in such a "conversational" mode, it turns off a line to the RS-232C port called Data Set Ready (DSR; pin 6). As long as DSR is off the computer will not try to use its DTR line to tell the modem to turn on its carrier tone.

The computer is also able to tell the modem whether or not to "take the receiver off the hook" in answer to an incoming call by using a signal line called Data Terminal Ready (DTR; pin 20). The computer can make this decision if it is too busy doing something else, if no one is home (i.e., there is no communications software running at the moment), or if it has been instructed to let only humans answer the phone.

An important third part of this DSR/DTR system for deciding on how to handle the phone is an RS-232C signal line called Ring Indicator (RI; pin 22). In a typical auto-answer system, the modem does not immediately answer the phone when it rings, rather it passes the ring signals along to the computer as pulses on the RI line. The computer may have been told to

wait for the sixth or seventh ring before answering, so nothing happens as long as you're around to grab the phone before the sixth ring. After that, the computer turns on DTR which causes the modem to answer. If the modem is set to conversational mode (DSR line off), your computer can now turn on a voice answering machine; if DSR is on, the computer can instead turn on RTS which causes the modem to send a carrier tone.



CD - Carrier Detect
CTS - Clear to Send (Modem has established carrier tone)
DCE - Data Communications Equipment (Modem)
DSR - Data Set Ready (Modem is prepared to generate tone)
DSRS - Data Signaling Rate Select (300 baud or 1200 bps)
DTE - Data Terminal Equipment (Computer or Terminal)
DTR - Data Terminal Ready (Computer is paying attention to Modem , may send commands to smart modem via TD line)
RD - Receive Data (Asynchronous Serial bits reassembled from demodulated signal)
RI - Ring Indicator
RTS - Request to Send (tells Modem to turn on carrier tone)
TD - Transmit Data (Asynchronous Serial bits sent to Modem for modulation and transmission)

All this makes for a pretty creditable computer/telephone interaction system (see Figure 17.7 for an overview). Several modems on the market have the facilities for all this, including the very inexpensive J-CAT 300 baud "portable" modem (it's just about 5 × 2 × 1 inch, so it sort of fits in your hand—$160.00) and the 212 AutoCAT ($770) 1200 bps modem, both from Novation.

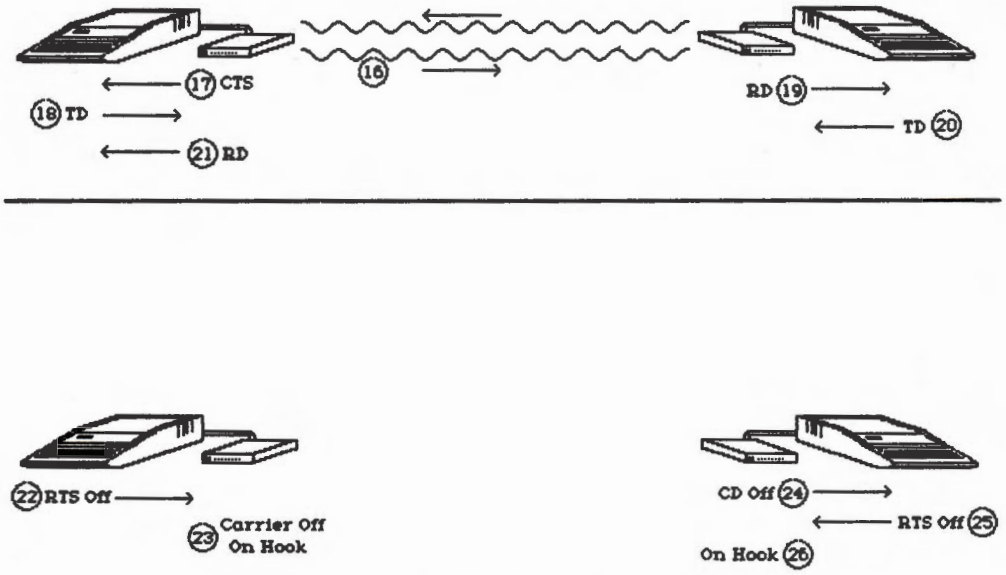


Fig. 17.7 RS-232 modem control and operation sequence.

1 and 2	Both modems are turned on.
3,4,5	Orginating computer indicates contact with its own modem and causes it to dial.
6,7,8	The answering computer learns that a call has come in, and activates its communication link with its own modem.
9,10,11,12	The answering computer orders its modem to generate a carrier tone at a selected rate. The modem does so and reports this back to its computer.
13, 14, 15	The orginating modem alerts its computer that it has received a carrier tone from another modem and its computer orders it to generate and send its own carrier.
16, 17	The orginating modem sets up its carrier and reports this fact to its computer.
18,19,20,21	Data communication takes place between the two computers.
22,23	The orginating computer orders its modem to turn off its carrier.
24,25,26	The answering modem reports loss of a carrier to its computer, and so it is told to turn off its carrier and "hang up the phone."

Smart Modems

You may be wondering: if you can put absolutely any information into the data stream for transmission, why must you rely upon this sort of Dark Ages system of turning on and off individual RS-232C wires to pass a tiny number of limited commands and status reports back and forth between the modem and the computer? No good reason except that this system was, in fact, worked out in the dark ages (sometime in the mid 1950s).

Dennis Hayes changed all that with the introduction of the Micromodem II, an internal modem card for the Apple, and then with the development of the Smartmodem 300, the first true external "smart modem." Modem cards for installation in Apple slots are "smart" because they are operated directly by the computer with the full facilities of the 50 signal lines on the Apple peripheral slots, and direct real time supervision by the 6502 and its memory resources.

The Smartmodem is "smart" because it has its own microprocessor and memory resources. The key functional difference between a smart modem and an average modem is the addition of what is called a "local command state." In an average modem, all communication between the modem and the computer is conducted via RS-232C lines 4, 5, 6, 8, 20, and 22. The data is carried on lines 2 and 3, but the modem makes no attempt to examine the data; it just modulates it or demodulates it and passes it on as if it were so much meaningless electrical noise.

In a smart modem, however, you can invoke the local command state in which the modem's own microprocessor intercepts the data coming in on line 2, interprets it, and executes any number of commands. It is then able to report back to the computer on line 3. When everything is squared away, the microprocessor can stop paying attention to the information in the data line and just supervise its transmission and reception.

External Smart Modems versus Internal Modem Cards

External modems such as the Hayes Smartmodem 1200 (see Figure 17.3) are popular because they can be used with any computer from any manufacturer, so there is usually a large body of software available to take advantage of their features. This is especially important for CP/M users, but in general, most communications software written for the Apple or various Apple coprocessors will be capable of operating an external Hayes modem. If you buy another kind of external modem you must be careful to find out if it is compatible with Hayes on the software level. If it is not, don't buy it unless the modem company can provide you with software which handles all of your needs.

Self contained modem cards for the Apple II, II+ and //e simplify your task in buying and setting up a modem system since you don't need a separate serial communications card. Further, they usually provide a variety of very sophisticated communication features which just can't be put into an external modem. Software for Apple modem cards is also dominated by Hayes (the Micromodem II; see Figure 17.10).

The modem cards from other manufacturers such as Multi-Tech and Novation (see Figure 17.4) typically offer more features. These modems are interesting because of their "bells and whistles." However, most commercial communications software won't let you do anything that a Hayes Micromodem II can't do, so all the nifty features will go unused. One important exception is that you may be able to make your Micromodem-compatible software run at 1200 bps if you use it with the Era 2 1200 bps modem card from Microcom.

External Smart Modems

In the Smartmodem 300 ($289), Hayes essentially took all of the features of the standard auto-dial/auto-answer modem and caused them to be executed by ASCII coded commands instead of by turning RS-232C signals on and off. This step alone greatly simplified modem operations for programmers and hardware designers. However, each command was provided with several additional options, and the modem was able to carry on many of the tasks while no program was running in the computer.

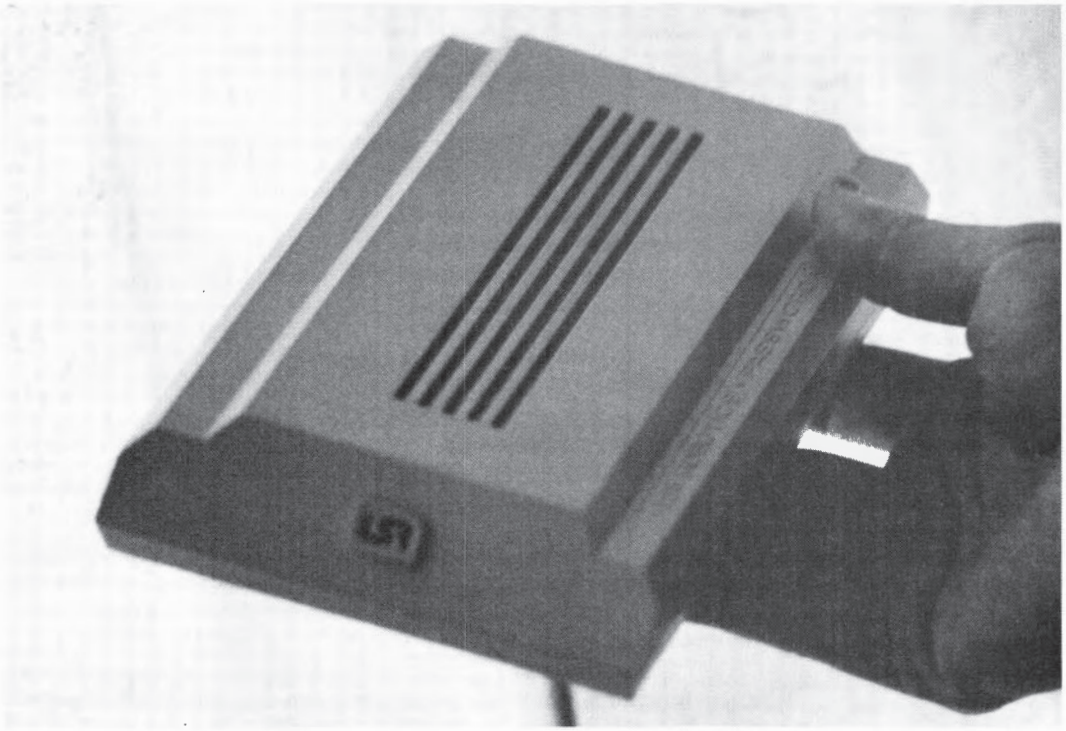


Fig. 17.8 Password 1200 bps modem from U.S. Robotics.

The result of all this is a much more versatile and easily operated device. Other features of the Smartmodem 300 include the ability to do touch tone dialing as well as "pulse" dialing (equivalent to standard turning the dial with your finger). This device was also popular with heavy industrial users because it provided simple control over a variety of modem features which just couldn't be altered on other modems.

The introduction of a 1200 bps version, the Smartmodem 1200 ($700) with essentially the same command structure placed the Hayes Smartmodem at the center of a great deal of the software development efforts of the past few years. This is why it is not wise to buy a smart modem from another company unless it is Hayes compatible at the software level, no matter how many extra features it offers. The Password modem (see Figure 17.7) uses more advanced electronics to duplicate most of the functions of, and attain full software compatibility with, the Smartmodem 1200, and still come in at a substantially lower cost. The Apple 1200 bps modem (see Figure 17.9) is based on a design similar to that used in the Password.

Fig. 17.9 Applemodem, a 1200 bps smart modem from Apple.

The Multi-Tech 212AD Intelligent Modem ($695) is an example of the current level of enhanced modems available from other manufacturers. Multi-tech claims that the 212AD can be configured for complete software compatibility with the Hayes Smartmodem 1200. It also provides several very useful additional features. When it dials a number, it is capable of detecting a busy signal. When a busy signal is encountered it can wait and redial the number, or it can select from among six different telephone numbers in its own memory, each up to 31 digits in length (for Sprint and MCI). It has a switch which lets you connect a regular hand set for voice communications, and it can be used with "synchronous" as well as asynchronous RS-232C and RS-422 systems (see Chapter 20). A second 212 compatible modem from Multi-Tech, the 212AH ($545) is software compatible with the Hayes Smartmodem 1200, but lacks most of the advanced features of the 212AD.

Another interesting entry in the 1200 bps smart modem marketplace is the Pro-Modem from Prometheus Products. The Pro-Modem provides all the features of a Hayes Smartmodem and has full software compatibility. The unique enhancement is the inclusion of a real time clock which can be addressed by the microprocessor inside the modem. The modem can be instructed to activate at a predetermined time, as well as to time stamp incoming files during unattended operation. The time is always available to ProDOS or to any program running in the Apple, whether or not the modem is being used for communication. At $495, this combined modem/clock is a fairly good buy, particularly for //c owners who need a clock.

The Transmodem 1200 from SSM is not compatible with the large body of commercial software written for Hayes, so you are limited to communications software from SSM. Its features are similar to the Hayes Smartmodem 1200 except that it can store and redial one 32 digit number

for MCI and Sprint. SSM sells the Transmodem in a package including one of their interface cards (such as the AIO-II) and a series of "Transend" software packages, which are fairly popular.

Modem cards for the Apple

The Micromodem II ($379) from Hayes serves conveniently as the standard with which the various other modem cards can be compared. It is essentially an auto-dial/auto-answer 300 baud modem with a built-in serial interface (6850 ACIA). A cable runs from the card to the outside of the Apple where a separate box contains the actual direct connect electronics and a receptacle for a modular telephone jack. It cannot do touch tone dialing so is difficult to use with Sprint or MCI.

When the Hayes Micromodem II was first released it gained immediate acceptance because of the variety of clever and useful programs and programming hints provided with it. During the subsequent five years as the most popular Apple modem, dozens of programmers have written excellent software for it. The software sold with the Micromodem from Hayes can work with DOS, CP/M or Pascal operating systems, so there are no rigid limits on your software or programming options. If, however, you're not happy with the Hayes software provided with the Micromodem, Hayes will gladly provide you with their Communications Software Directory which lists dozens of compatible communications software packages from third party sources including software for transmitting high res graphics images, for interactive work on VisiCalc models, and many more.

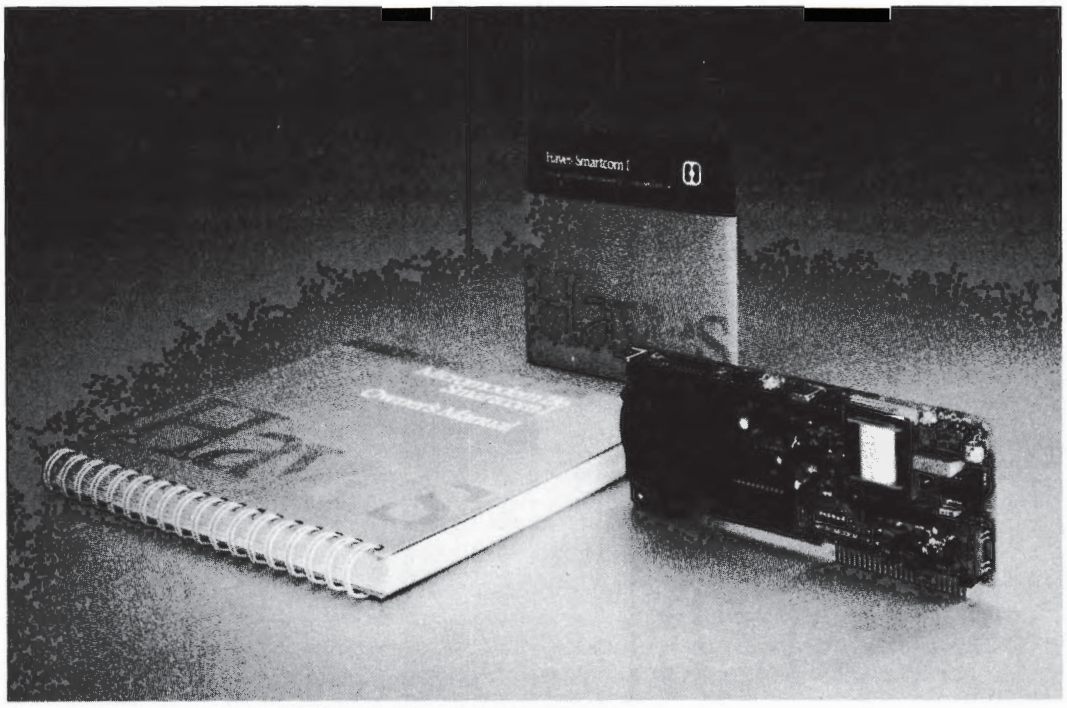


Fig. 17.10 Hayes Micromodem //e.

An important new arrival is the Hayes Micromodem //e (see Figure 17.10), a 1983 update of the classic modem card. It adds automatic touch tone dialing and an attached speaker to hear if you've gotten a busy signal, a human, or a phone company message. The contents of the external coupler box have shrunk and been placed on the card itself. Hayes provides a modular telephone jack receptacle which attaches to the back panel of the computer which makes connecting and disconnecting very convenient. The new owner's manual is also an amazing thing—the inclusion of several dozen color photographs along with table charts and teaching material is certainly without parallel in the microcomputer industry.

The Modemcard from SSM ($299) is similar to the Micromodem //e in that it provides touch tone dialing and requires no external box. The modular jack receptacle is mounted directly on the card so there are no cables at all other than your regular telephone line, and SSM claims complete software compatibility with the Hayes Micromodem. This is a reasonable alternative and it includes a free subscription to The Source, a popular modem information service. The documentation, unfortunately, retains SSM's traditional orientation towards programmers rather than towards the non-programming user.

The Multi-Tech Modem II ($399) is quite similar to the original Micromodem II except for the addition of some status lights on the external direct coupler box and the provision of an additional jack for attachment of a regular handset. An updated Modem //e from Multi-Tech does not require an external coupler box.

The Operator from Timecor ($169) is certainly the least expensive modem card, but despite what the ads say the Operator cannot auto-dial either pulse or touch tone. This is a limited feature modem and although Timecor claims software compatibility with the Micromodem, some software options won't work because the hardware is not fully compatible.

Advanced Feature Modem Cards

There are few single card, 1200 bps, 212 compatible modemcards. The ERA 2 from Microcom was the first to come to market, so it set new standards of its own. This modem is designed to be compatible with software written for the Hayes Micromodem, but to allow the user to operate at 1200 bps (even on the //e) and provide software compatibility with the Smartmodem 1200 as well. The ERA 2 comes with a high accuracy data transmission protocol system (called MNP) which can be used with other Microcom products for the IBM PC, DEC VAX, TRS 80, etc., to assure near 100 percent reliability of data transfers. In addition, the ERA 2 comes with terminal emulation software to let your Apple pass itself off as if it were an IBM 3101, DEC VT-100, or VT-52 terminal.

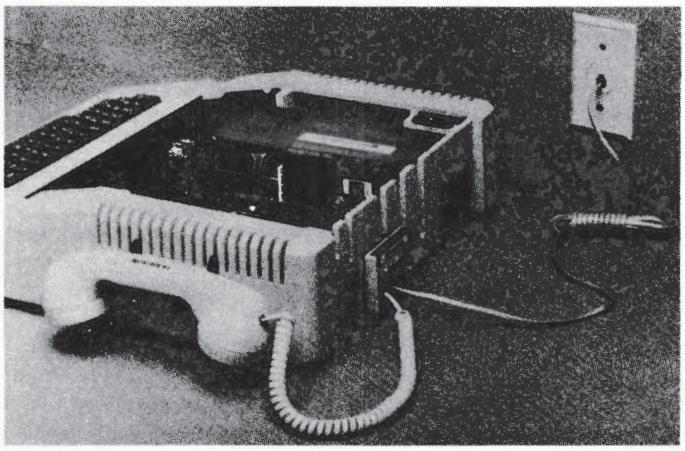


Fig. 17.11 Novation Apple-Cat II w/accessories module.

The Apple Cat II system from Novation is really in quite a different class from any of the other Apple modem cards or external modems. Although the minimal configuration of the Apple CAT ($389) is roughly similar in features to the Micromodem //e, the various enhancements from Novation make this a very remarkable and interesting product. With the expansion module attached to the back panel of the Apple, the Apple CAT does double duty as a printer interface card, and it provides a BSR interface (see Chapter 14) which can send controlling signals through your home or building's AC power lines to operate lights, air conditioners, electric door locks, etc.

The Apple Cat II can be configured to act as a normal voice telephone with the handset plugged into the expansion module (see Figure 17.11), and if you call the modem from a distant location, you can get it on line and then use the buttons on a touch tone phone to send itmessages—i.e., tell it to turn off lights in your house, order it to make a phone call and send a file to another modem, etc. Novation provides a disk with its Com Ware software for driving these features as well as for getting the time of day from a clock card elsewhere in the Apple for work scheduled by time of day. Novation offers an optional ROM chip containing Com Ware II so that no program needs to be loaded from disk. Finally, a second card, the 212 upgrade (see Figure 17.12), provides all of the previously mentioned features as well as 1200 bps high speed communication. The complete 212-Apple CAT II system costs nearly $900, but when fully installed it really makes your Apple work for its supper!

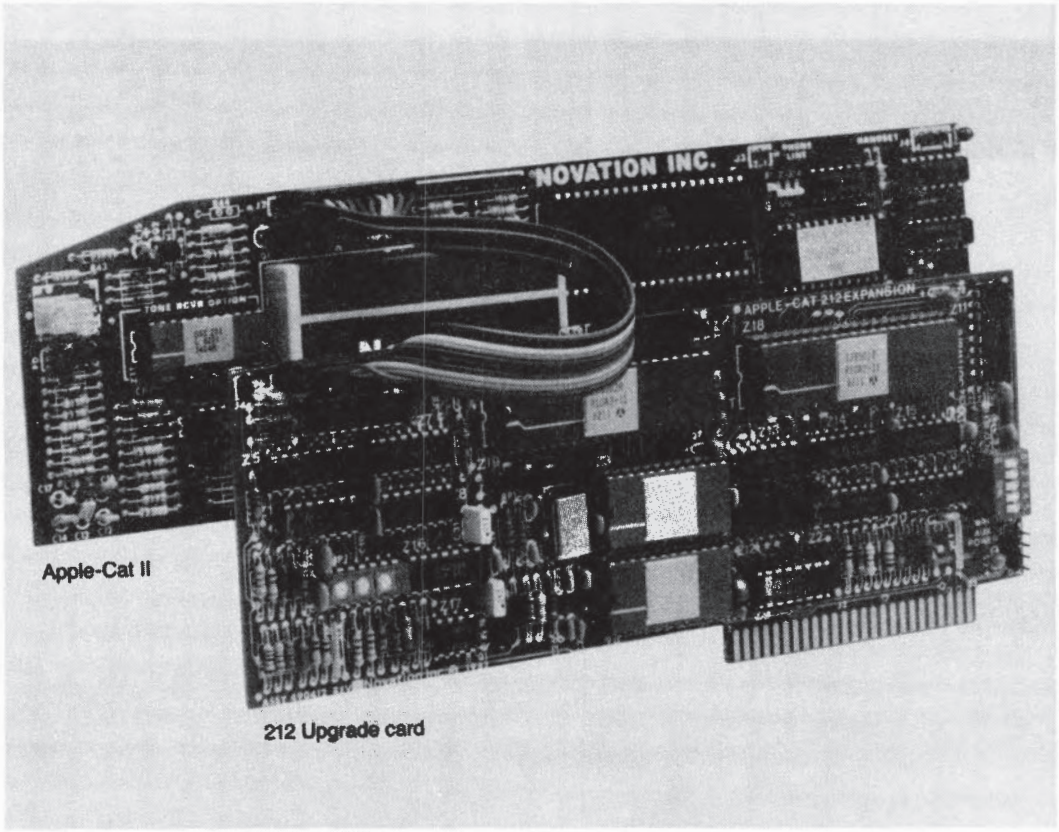
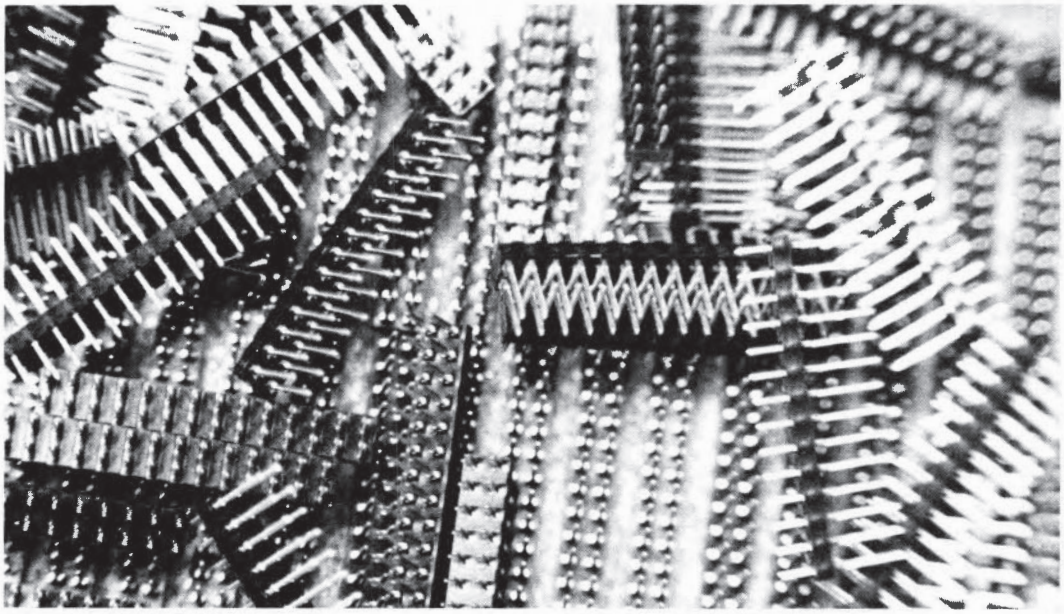


Fig. 17.12 Apple-Cat II w/212 upgrade for 1200 bps communications.



Chapter 18

Interfaces for Terminals, Modems and Printers

Communicating with Video Boards and Terminals

The Apple //e 80 column text card is designed to behave in some ways as if it were an external CRT terminal made by Datamedia. The trend toward mimicking Datamedia terminals has a long history among the various Apple 80 column boards such as the Videoterm from Videx and the Smarterm board from ALS. This mimicking takes place in a very limited way, at the level of communications between the computer and the display system.

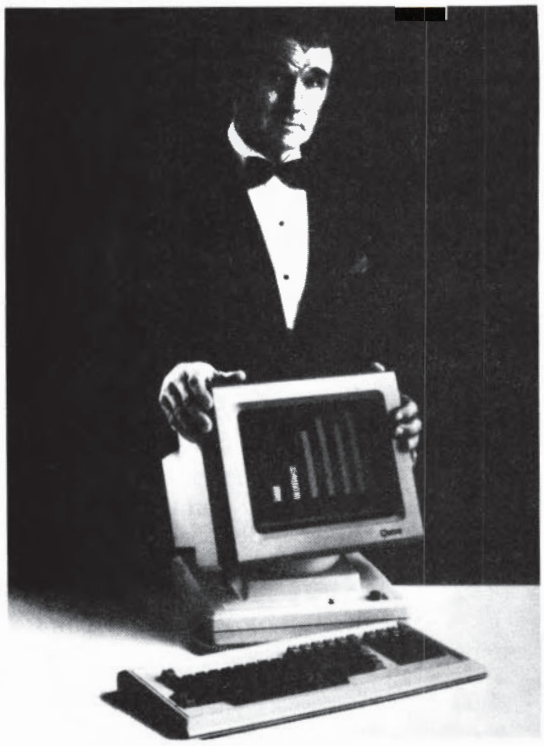


Fig. 18.1 Qume QVT-102 CRT terminal. Eighty column cards intended for slot 3 in the Apple carry most of the electronics of such a terminal. The //e 80 column card system is different electronically, but shares many of the same software features.

To clarify a bit, it's handy to have in mind both an 80 column card (see Chapter 6, Figure 6.4) and the CRT terminal in Figure 18.1. The 80 column card is roughly equivalent to the sum total of the digital circuitry inside the CRT terminal, and it is operated in a somewhat similar way.

The design of terminal electronics assumes that a computer will be sending characters to the terminal via an RS-232C connection. Unlike the modem connection systems described in Chapter 17, the RS-232C connections between a computer and a terminal are usually extremely simple. Only the two signal lines, Receive Data (RD) and Transmit Data (TD) are used, and communication is strictly limited to the passage of ASCII codes in asynchronous serial format (see Chapter 16).

The standard ASCII codes are quite adequate for sending letters and numbers from a computer to a terminal for display. However, there are some inadequacies. The computer must be able to order the terminal to do a few other tricks.

Among the necessary additional tricks are clearing the screen, moving the cursor from place to place, changing to inverse, etc. In fact, modern CRT terminals typically offer dozens of these screen management and cursor control functions. However, the only means available to the computer for telling the terminal to do these things is the passage of ASCII codes. This requires that the terminal examine each incoming code and decide whether it is supposed to pick the appropriate letter or number and display it on the screen or respond to the code by changing its operation in some way.

Control Codes and Escape Sequences

The original ASCII system included 32 different "control codes" for this purpose, but many manufacturers of terminals, as well as manufacturers of printers, modems and buffers, have supplemented the original control codes with a much larger number of two and three key "escape sequences."

If you look at Table 18.1 you can see the positions of these control codes in the ASCII table. However, although these control codes were made available for sending controlling messages between the computer and the terminal, there has never been any agreement among terminal manufacturers about which code is supposed to control which function.

For instance, the ASCII code you get by holding down the "control" key and then pressing "L" is code number 12, (CTRL-L) and the designers of the code called this the Form Feed (FF) function. This was meant as an instruction to a printing terminal that it should roll a new sheet of paper into place. Great, but since CRT terminals do not have rolls of paper streaming through, what does it mean when they're told to do a form feed?

If you send a "CTRL-L" to a Datamedia type terminal, the cursor pops up to the top left corner and the screen clears. This seems like a reasonable interpretation of a form feed on a CRT screen. But the makers of the very popular Soroc IQ 120 terminal decided that the idea of a form feed was just inappropriate, so when a Soroc terminal sees a CTRL-L it responds by moving the cursor forward one space. This was a totally arbitrary choice of function, and it is one of 25 or 30 such arbitrary assignments made by this manufacturer. Other manufacturers made their own equally arbitrary and equally incompatible assignments.

For various historical reasons, most of the CP/M programs used by Apple owners (i.e., WordStar) seem to want to act as if they were speaking to a Soroc IQ 120. If these programs were left

ASCII Code Table

Bits 4,3,2,1,0	Bits 7,6,5 = 000			Bits 7,6,5 = 001			Bits 7,6,5 = 010			Bits 7,6,5 = 011		
	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
0 0000	NUL	00	00	SP	32	20	@	64	40	`	96	60
0 0001	[CTRL A] SOH	01	01	!	33	21	A	65	41	a	97	61
0 0010	[CTRL B] STX	02	02	"	34	22	B	66	42	b	98	62
0 0011	[CTRL C] ETX	03	03	#	35	23	C	67	43	c	99	63
0 0100	[CTRL D] EOT	04	04	$	36	24	D	68	44	d	100	64
0 0101	[CTRL E] ENQ	05	05	%	37	25	E	69	45	e	101	65
0 0110	[CTRL F] ACK	06	06	&	38	26	F	70	46	f	102	66
0 0111	[CTRL G] BEL	07	07	'	39	27	G	71	47	g	103	67
0 1000	[CTRL H] BS	08	08	(	40	28	H	72	48	h	104	68
0 1001	[CTRL I] HT	09	09	)	41	29	I	73	49	i	105	69
0 1010	[CTRL J] LF	10	0A	*	42	2A	J	74	4A	j	106	6A
0 1011	[CTRL K] VT	11	0B	+	43	2B	K	75	4B	k	107	6B
0 1100	[CTRL L] FF	12	0C	,	44	2C	L	76	4C	l	108	6C
0 1101	[CTRL M] CR	13	0D	-	45	2D	M	77	4D	m	109	6D
0 1110	[CTRL N] SO	14	0E	.	46	2E	N	78	4E	n	110	6E
0 1111	[CTRL O] SI	15	0F	/	47	2F	O	79	4F	o	111	6F
1 0000	[CTRL P] DLE	16	10	0	48	30	P	80	50	p	112	70
1 0001	[CTRL Q] DC1	17	11	1	49	31	Q	81	51	q	113	71
1 0010	[CTRL R] DC2	18	12	2	50	32	R	82	52	r	114	72
1 0011	[CTRL S] DC3	19	13	3	51	33	S	83	53	s	115	73
1 0100	[CTRL T] DC4	20	14	4	52	34	T	84	54	t	116	74
1 0101	[CTRL U] NAK	21	15	5	53	35	U	85	55	u	117	75
1 0110	[CTRL V] SYN	22	16	6	54	36	V	86	56	v	118	76
1 0111	[CTRL W] ETB	23	17	7	55	37	W	87	57	w	119	77
1 1000	[CTRL X] CAN	24	18	8	56	38	X	88	58	x	120	78
1 1001	[CTRL Y] EM	25	19	9	57	39	Y	89	59	y	121	79
1 1010	[CTRL Z] SUB	26	1A	:	58	3A	Z	90	5A	z	122	7A
1 1011	ESC	27	1B	;	59	3B	[	91	5B	{	123	7B
1 1100	[CTRL \] FS	28	1C	<	60	3C	\	92	5C	|	124	7C
1 1101	[CTRL]] GS	29	1D	=	61	3D	]	93	5D	}	125	7D
1 1110	[CTRL ^] RS	30	1E	>	62	3E	^	94	5E	~	126	7E
1 1111	[CTRL _] US	31	1F	?	63	3F	_	95	5F	DEL	127	7F

ACK - Acknowledge
BEL - Bell
BS - Backspace
CAN - Cancel
CR - Carriage Return
DC1 - Device Control 1 (XON)
DC2 - Device Control 2
DC3 - Device Control 3 (XOFF)
DC4 - Device Control 4
DEL - Delete
DLE - Data Link Escape
EM - End of Medium
ENQ - Enquiry
EOT - End of Transmission
ESC - Escape
ETB - End of Transmission Block
ETX - End of Text

FF - Form Feed
FS - File Separator
GS - Group Separator
HT - Horizontal Tab
LF - Line Feed
NAK - Negative Acknowledge
NUL - Null
RS - Record Separator
SI - Shift In
SO - Shift Out
SOH - Start of Heading
STX - Start of Text
SUB - Substitute
SYN - Synchronous Idle
US - Unit Separator
VT - Vertical Tab

Table 18.1 ASCII Code Table.

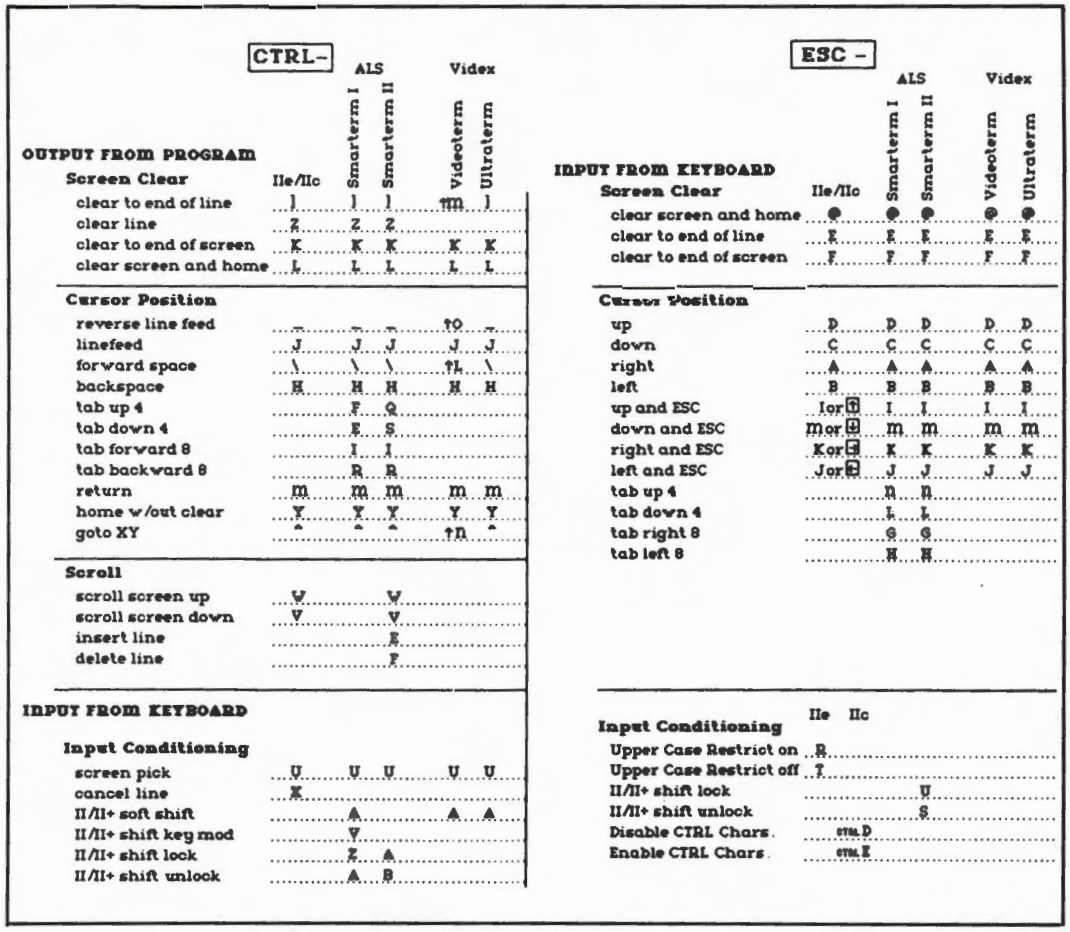
to their own devices, the Apple's Datamedia-type cursor would zoom from place to place at inappropriate times, text would disappear erratically, etc. To prevent this, all Apple CP/M operating systems intercept every one of the ASCII codes coming from a program before passing them along to the 80 column card.

If a given code represents a letter or number, it is passed on immediately for display. If, however, CP/M discovers that the code is a Soroc screen control function, it halts that code's

passage, stops to look up which code will cause the same function in a Datamedia terminal, and then passes that Datamedia code to the 80 column card instead of what the program was trying to send. The code that arrives at the screen is different from the code which was sent by the program, but the screen does what the program wanted it to. Table 18.2 lists the control codes used by the Apple //e 80 column card as well as those for the Videoterm, the Ultraterm, and the Smarterm I.

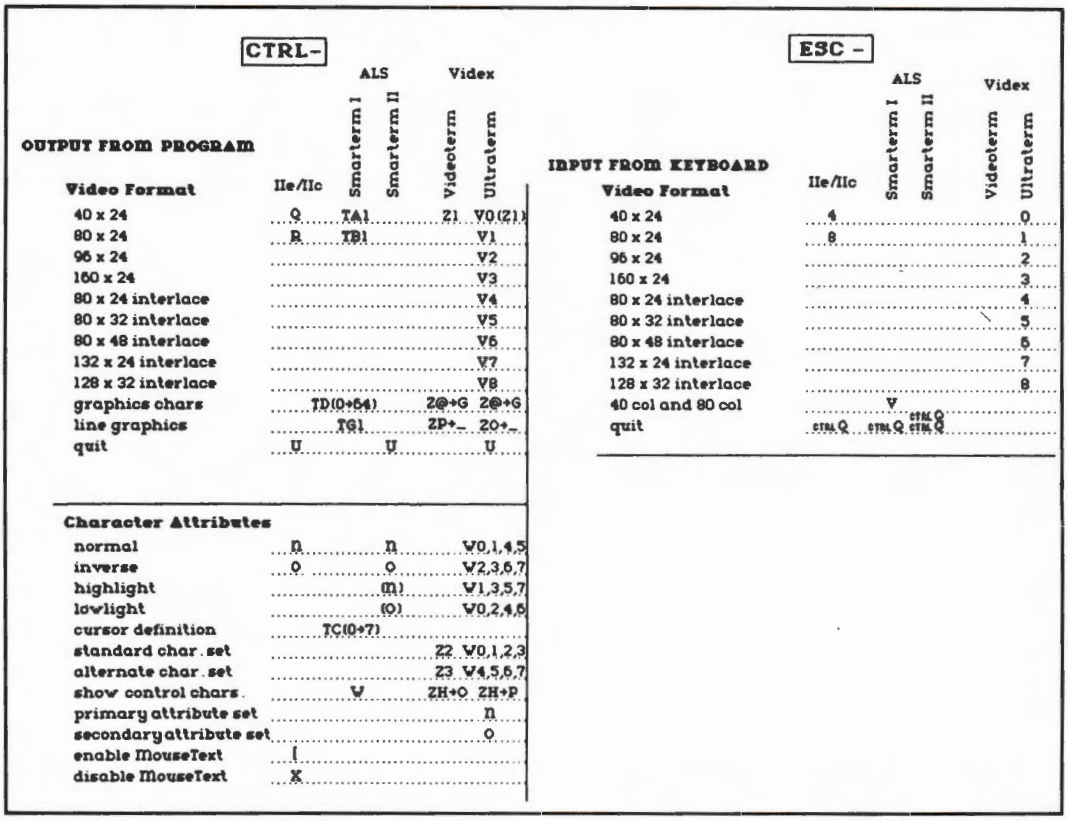
This system for encoding control operations actually gets even more complicated, because the 32 available control codes just aren't sufficient to provide for all the features that terminal manufacturers like to include. Control codes are therefore supplemented by the escape sequences. There are a potentially unlimited number of escape sequences because they are encoded differently from control codes.

The 32 control codes are real, actual and fixed ASCII symbols. Escape codes, however, are always made up of at least two characters. The first of these is usually ESC, the "escape control code" (ASCII 27 in Table 18.1). When most terminals receive ASCII code 27 they react by treating the next one or two ASCII codes in a special way. These escape sequences (i.e., Escape A) are completely non-standard among terminal manufacturers and also require translation when sent from one kind of program to another kind of terminal, and Apple CP/M performs some of these translations.



CTRL-	IIe/IIc	ALS Smarterm I	ALS Smarterm II	Videx Videoterm	Videx Ultraterm
OUTPUT FROM PROGRAM					
Screen Clear					
clear to end of line	]	]	]	^m	]
clear line	Z	Z	Z		
clear to end of screen	K	K	K	K	K
clear screen and home	L	L	L	L	L
Cursor Position					
reverse line feed	_	_	_	^O	_
linefeed	J	J	J	J	J
forward space	\	\	\	^L	\
backspace	H	H	H	H	H
tab up 4		F	Q		
tab down 4		E	S		
tab forward 8		I	I		
tab backward 8		R	R		
return	m	m	m	m	m
home w/out clear	^	^	^	^	^
goto XY				^n	
Scroll					
scroll screen up	V		V		
scroll screen down	V		V		
insert line			E		
delete line			F		
INPUT FROM KEYBOARD					
Input Conditioning					
screen pick	U	U	U	U	U
cancel line	X				
II/II+ soft shift		A		A	A
II/II+ shift key mod		V			
II/II+ shift lock		Z	A		
II/II+ shift unlock		A	B		

ESC -	IIe/IIc	ALS Smarterm I	ALS Smarterm II	Videx Videoterm	Videx Ultraterm
INPUT FROM KEYBOARD					
Screen Clear					
clear screen and home	●	●	●	●	●
clear to end of line	E	E	E	E	E
clear to end of screen	F	F	F	F	F
Cursor Position					
up	P	P	P	P	P
down	C	C	C	C	C
right	A	A	A	A	A
left	B	B	B	B	B
up and ESC	I or ↑	I	I	I	I
down and ESC	M or ↓	M	M	M	M
right and ESC	K or →	K	K	K	K
left and ESC	J or ←	J	J	J	J
tab up 4		N	N		
tab down 4		L	L		
tab right 8		G	G		
tab left 8		H	H		
Input Conditioning	IIe	IIc			
Upper Case Restrict on	R				
Upper Case Restrict off	T				
II/II+ shift lock		U			
II/II+ shift unlock		S			
Disable CTRL Chars.	esc D				
Enable CTRL Chars.	esc E				



OUTPUT FROM PROGRAM — CTRL-

Video Format	IIe/IIc	ALS Smarterm I	ALS Smarterm II	Videx Videoterm	Videx Ultraterm
40 x 24	Q	TA1		Z1	V0(Z1)
80 x 24	R	TB1			V1
96 x 24					V2
160 x 24					V3
80 x 24 interlace					V4
80 x 32 interlace					V5
80 x 48 interlace					V6
132 x 24 interlace					V7
128 x 32 interlace					V8
graphics chars		TD(0+64)		Z@+G	Z@+G
line graphics		TG1		ZP+_	ZO+_
quit	U		U		U

IMPUT FROM KEYBOARD — ESC -

Video Format	IIe/IIc	ALS Smarterm I	ALS Smarterm II	Videx Videoterm	Videx Ultraterm
40 x 24	4				0
80 x 24	8				1
96 x 24					2
160 x 24					3
80 x 24 interlace					4
80 x 32 interlace					5
80 x 48 interlace					6
132 x 24 interlace					7
128 x 32 interlace					8
40 col and 80 col		V			
quit	CTRL Q	CTRL Q	CTRL Q CTRL Q		

Character Attributes

normal	N		N	V0,1,4,5
inverse	O		O	V2,3,6,7
highlight			(1)	V1,3,5,7
lowlight			(0)	V0,2,4,6
cursor definition		TC(0+7)		
standard char. set				Z2 V0,1,2,3
alternate char. set				Z3 V4,5,6,7
show control chars.		V		ZH+O ZH+P
primary attribute set				N
secondary attribute set				O
enable MouseText	I			
disable MouseText	X			

Table 18.2 Video terminal card Escape and Control codes.

Apple Screen Escape versus Apple Screen Control

In the Apple itself there is a subtle but important distinction between the use of control codes and escape sequences. The escape sequences are interpreted by the keyboard input system which then sends appropriate groups of control codes to the 80 column card to produce the function requested by the escape sequence. This means that you cannot make the escape sequences work when they are "output" (by a PRINT statement) in a BASIC program. They only work when they are typed at the keyboard and interpreted by the keyboard input system.

The control codes are interpreted by the screen management system, so they will work when output from a program. These control codes also work when typed at the keyboard because the keyboard input system simply takes them in and then outputs them to the screen system, at which time they take effect.

The result of this difference between escape sequences and control codes in the Apple is that Apple DOS software never includes escape sequences for the screen, but only uses the fairly standard Datamedia control codes. Table 18.2 lists the escape sequences for the //e 80 column text card, for the Videoterm, for the Ultraterm, and for the Smarterm.

Escape Sequences for Printers, Modems and Buffers

Modern dot matrix and letter quality printers make very extensive use of escape sequences. This is how word processing and other programs control boldface, proportional spacing, tabs, margins, etc. Escape sequences embedded in the ASCII data stream cause the printer to take some special action other than just printing.

You may spend weeks reading literature on printers before making your choice based on speed and quality of character formation only to discover that letters, numbers, carriage returns and line feeds are all you will ever get your printer to respond to. Escape sequences among printers not too well standardized, so you can guide your purchase only by choosing a popular and well known printer which your word processor or graphics software knows how to operate. Newer or less popular printers should only be purchased if the manufacturer can guarantee substantial escape code compatibility with some very popular and well known older printer, such as the Diablo 630 or Qume Sprint 5 (see Chapter 19, Table 19.1).

The major innovation underlying the emergence of smart modems is the use of a local command state which is invoked by an escape sequence. Modems cannot use ASCII code 27 (ESC) as the escape sequence lead-in character because they must be able to pass "ESC" on down the line for use by printers and CRT terminals. Hayes chose "+ + +" following a pause as the way of indicating to the modem that the next few ASCII codes have special meaning. Other modems may actually use the same command structure as Hayes, but not actually be compatible because they use a different "escape" lead-in character.

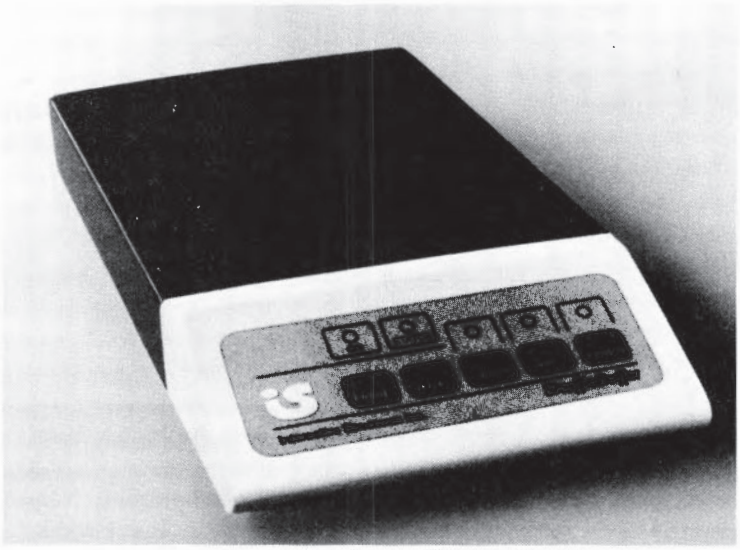


Fig. 18.2 Shuffle Buffer from Interactive Structures. This is a smart buffer which will touch off a new flowering of printer buffer escape codes.

The first "smart" printer buffer appeared in early 1984 (the Shuffle Buffer from Interactive Structures; see Figure 18.2) and it uses the "@" sign as an escape sequence lead-in character to let the computer embed instructions within the ASCII data stream.

Serial and Parallel Interface Protocols

There are lots of small particulars to keep in mind about control signals for modems (see Chapter 17), nonetheless, when you want to connect a modem to a computer, your task is extremely simple. You always use an RS-232C connector (a DB-25), and every time you do this, no matter whose modem you use, it will work. Line 2 will always be Transmit Data and line 4 will always be Request To Send.

If you suspect printer interfacing is that simple, get ready for a shock. There are literally dozens of different connection systems. There are at least six different types of connectors in use, and even if your interface card and your printer have connectors which look physically identical, there is absolutely no guarantee that the same wires will be connected to the same pins at each end.

Finally, and most devastating, even if the connectors are the same and the wires are connected to the same pins on both ends, the timing and voltage levels may make the two incompatible. There are actually computer owners out there who had an interface card and a printer which both claimed to be "Centronics Standard" (one of the most popular printer interfacing system), but got smoke coming up out of their interface card when they plugged it all in (the "Centronics pin 14 problem," for the afficianados in the crowd). This is an inexcusable zoo.

What, you may ask, is wrong with these people? Why can't printer designers get a hold of themselves and sit down and work this thing out once and for all. Good question! Why can't they? The good news is that most computer owners have taken action by refusing to buy printers from any company other than Epson or Leading Edge (C. Itoh), which always use exactly the same kind of Centronics standard parallel connection system. So there is some justice.

Serial Handshake Problems

The problem grows out of the early days when very few printers were being used with word processing programs. Back then, the principal use for printers was to type out "listings" of computer programs to give programmers "hardcopy" to look at. The divorce between printers and CRT terminals was recent, and printer designers naturally chose to try to use the RS-232C interface system described in detail above (in Chapter 17).

There is, however, a very fundamental and simple difference between the way a printer handles characters and the way they are handled by a CRT terminal, modem or computer. The RS-232C standard was set up to include a variety of control wires. The purpose of these wires was to help the computer and the modem work together to set up a telephone link with some other modem/computer pair. Once the link is established, RS-232C assumes that data can flow freely at a steady baud rate. There is no provision in the RS-232C system for examining the contents of the data stream. It just sets up the link.

Printers, however, are like typewriters. They do not just zip along continuously. If you listen to a fast typist you will hear a steady rat-a-tata-tata until the end of the line arrives. A bell rings, the typist reaches for the Return key, and then there is a break in the rhythm while the print head zooms back to the left side (the carriage return) and the platten is rolled up one line (the line feed), then the rat-a-tata-tata resumes, punctuated regularly by pauses for carriage returns. After a while the typist gets to the bottom of the page and there is a much longer pause while the page is removed and a new piece of paper is rolled into place (the form feed).

(The preceeding paragraph was directed at all of you folks who were brought up on word processors and have never actually seen or heard a typewriter being used.)

But seriously, the point is that some characters must be followed by long pauses to allow the print head to swing across the page or to allow the next page to roll into place. If you use a simple RS-232C system, the characters will just keep flowing to the printer in a steady stream,

and all the ones that come along during a carriage return or a form feed will be lost and never get printed.

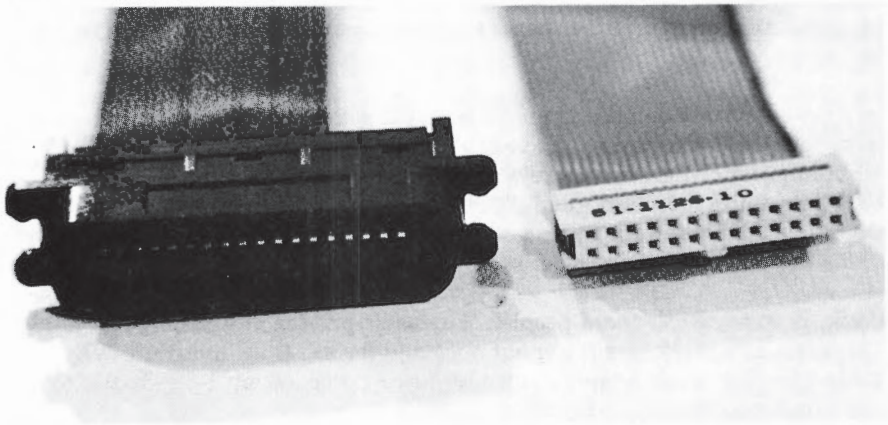


Fig. 18.3 The Amphenol 57-30360 connector on the left is used in Centronics Standard interfaces to plug into the printer. The 26 pin header connector on the right is used in a large number of different interface systems to plug directly into electronic circuit boards.

There are two kinds of solutions to this problem. One is a software solution. Just keep track of when you've sent out the last character on a line, and then wait a little bit before sending out the next character. The Apple Super Serial Card is set up to accommodate this sort of solution in that it accepts instructions on how many characters will be printed in each line and how long it should pause after each carriage return and after each form feed. However, it does not know when it is supposed to cause a form feed and it can get in trouble if the form feed is triggered automatically by the printer. Of course, any word processing program worth its salt knows exactly when there will be a carriage return or form feed and can take care of inserting pauses.

This all might have had a happy ending except that this sort of stuff requires a fairly sophisticated printer interface card, or a word processor which has a fairly rock solid time base; two conditions which are rarely met.

Meanwhile, printer manufacturers came up with their own solution to this rhythm problem; which turned out to be no solution at all and made the situation rather worse. This solution was to include memory chips or "buffers" in the printers. The assumption was that the printer could accept a steady stream of data from a standard RS-232C system, and whenever a carriage return or form feed took place, the extra characters would pile up in the buffer to be printed a few moments later without being lost. Sounds great. But there is this one little flaw.

With a buffer built into the printer, the interface card and the word processor do not have to insert pauses after carriage returns and form feeds; however, what happens when the buffer gets full? If characters are flowing in at the same rate that they are being printed (not actually a good assumption) then with every carriage return and form feed a few more characters will pile up in the buffer. The buffer also fills up if the computer sends characters faster than the machine can type.

When the buffer is full, the computer must stop sending characters for a while. However, unlike the simple situation with carriage return and form feed pauses, the computer has no simple way of guessing in advance when some printer's buffer is going to be full.

All this points to the need for a printer to send signals to the computer to inform it as to whether or not it is OK to send a character at that particular moment. Now those of you who carefully worked your way through the sections on modem control by the RS-232C system are probably wondering why these devices cannot make use of CTS, RTS, DTR or DSR (clear to send, request to send, data terminal ready and data set ready, respectively). Well, there is a fatal flaw in the system.

The designers of the RS-232C standard assumed that no one would consider sending data until the control lines had already done their work to set up the telephone link. Similarly, they assumed that no one would intentionally try to disconnect the phone link while data was being transmitted. And so they left out one key element in the system. There is no link between the time that a control signal is changed and the time that a given character is completely sent or received. Nothing prevents the disconnect from occurring directly in the middle of a character. If this were allowed to happen while you were printing out a document, there would be a few garbled or absent characters scattered around in every printed document; not very professional to say the least.

The RDY/BSY Approach and its Variants

This flaw is not simple to correct because the RS-232C signals are built into the chip design of the ACIA 6850 and 6551 (see Chapter 16) and other serial interface devices. Somehow, there was never a sufficient consensus on the issue to see that anything got done about it. Programmers could get around the flaw with what is called the Ready/Busy (RDY/BSY) system. This just requires that the program check the interface card to see if the control signals have changed before it hands over a character, and this should be done before every single character.

Most serial interfaces for printers now being sold use some variant of this RDY/BSY system. The problem is that it is by no means clear which of RTS, CTS, DTR or DSR should be used by the printer to signal that its buffer is full. Most printers which are sold with serial RDY/BSY interfaces use DTR (pin 20) to signal when their buffer is full but, for instance, many NEC printers use pin 19, the Texas Instruments 810 uses pin 11, the Heathkit H14 uses pin 15, and the Olympia ES-100 uses pin 4.

Even if two printers both use DTR, there may be disagreement about whether they are supposed to turn it on when the buffer is full or whether they are supposed to turn it on when it is empty. This is called a difference in "polarity," so two printers may use the same connection and the same protocol and the same pin, but have different polarities. So when you are told that your printer has a serial RS-232C interface, you are dealing with at least one loose live wire until you find out which of its RS-232C pins it is using for RDY/BSY, which pin your interface card is expecting to see RDY/BSY on, and how the two of them feel about polarity.

The Transmit Data Approach and its Variants

Some printer manufacturers who became distressed about the variety of hardware connections for the RDY/BSY system tried a different approach based on the data lines rather than the control lines. These printers use the Transmit Data line to carry on conversations with the computer about the status of their buffers.

The most popular of these protocols is called XON/XOFF. When these printers sense that their buffer is about to be full, they use the TD line to send an XOFF signal (in the form of a special ASCII code) back to the computer warning it to stop sending characters as soon as possible. Once the buffer is nearly empty again, the printer sends a different ASCII code to the computer which instructs it to resume. The XOFF character for stopping is a CTRL-S (ASCII code 19 = Device Control 3; DC3) and the XON code to call for more data is CTRL Q (ASCII code 17 = Device Control 1; DC1).

A second of these TD based systems is called ETX/ACK. In this protocol, the computer sends one full line (80 characters) of data followed by an End of Text (ETX) code (ASCII code 3 = CTRL-C). The printer captures the line of data and goes about printing it, but when it when it gets to the end of line and sees the ETX code, it sends an Acknowledge (ACK) code (ASCII 6 = CTRL-F) back to the computer to announce that it has room for a new line of text.

The third system is called ENQ/ACK, and in this system, the computer initiates the exchange by sending an Enquiry (ENQ) character (ASCII 5 = CTRL-E) to the printer. If the printer has space in its buffer for a line of text, it responds with an "ACK" code which it sends along the TD line.

Parallel Protocols

While all this jockeying and fiddling was going on in pursuit of the ultimate serial printer interface, other manufacturers tried to start from scratch, abandon the metaphor of "printer as serial terminal," and switch to a completely different interface system. The principal alternative was to treat the printer as if it were on a sort of direct extension of the computer's data bus. All eight wires from the data bus would be extended directly out to the printer, and the microprocessor would take direct control of passing the ASCII codes to the printer on a character by character basis.

This sort of arrangement is called a "parallel" interface because the eight bits in each byte proceed to the printer all at once side-by-side in the eight wires. Data transmission is much simpler than in the RS-232C system because each character is sent as a single event rather than as a stream of 10 serial bits. The signals are all 0 volts or 5 volts just as in any standard TTL system (see Chapter 13), but this means that cable length is limited to two or three feet at the most rather than the 50 feet you can use with a serial RS-232C cable.

The various kinds of parallel printer interfaces all use the same system to control the flow of characters to the printer. An ASCII code is loaded into the interface in preparation for transmission, but nothing happens until the printer sends a brief pulse to the computer along a wire called Acknowledge "ACK" (this ACK wire is not related to the ASCII code ACK used in serial interfaces and discussed above; they probably used the same name just to throw people off). The printer therefore has complete control over when each character will arrive. There is no risk of overflowing buffers, lost characters, or characters chopped in half by a poorly timed busy signal.

When a parallel interface receives an ACK signal from a printer it loads the next character onto the eight data lines and then it sends a pulse to the printer on a tenth wire called Strobe (STB). The printer watches the strobe line and when it sees a pulse it assumes that a new ASCII code has arrived and may be "latched" in for printing or temporary storage.

This ACK/STB system is called "handshaking" because, unlike the various systems used for serial interfaces, both the printer and the computer are involved. For every character sent there is first an acknowledge from the printer and then a strobe from the computer.

Incompatibility Among Parallel Printer Interfaces

This 10 wire system of eight data lines and two control lines sounds so simple, straightforward and appropriate that you might think it would be impossible to design parallel printer interfaces to be incompatible with each other. In sad truth, the actual situation is as bad or worse than the situation among serial interfaces.

The differences among the parallel printer interfaces are very trivial but extremely difficult to deal with. The most fundamental problem is that at least three different connectors are used, a DB-25 connector for IDS parallel printers (which makes these look like RS-232C serial devices from the outside), a 3M 3464 connector for the Centronics 730 printer, and an Amphenol 57-30360 for the Centronics 779 printer.

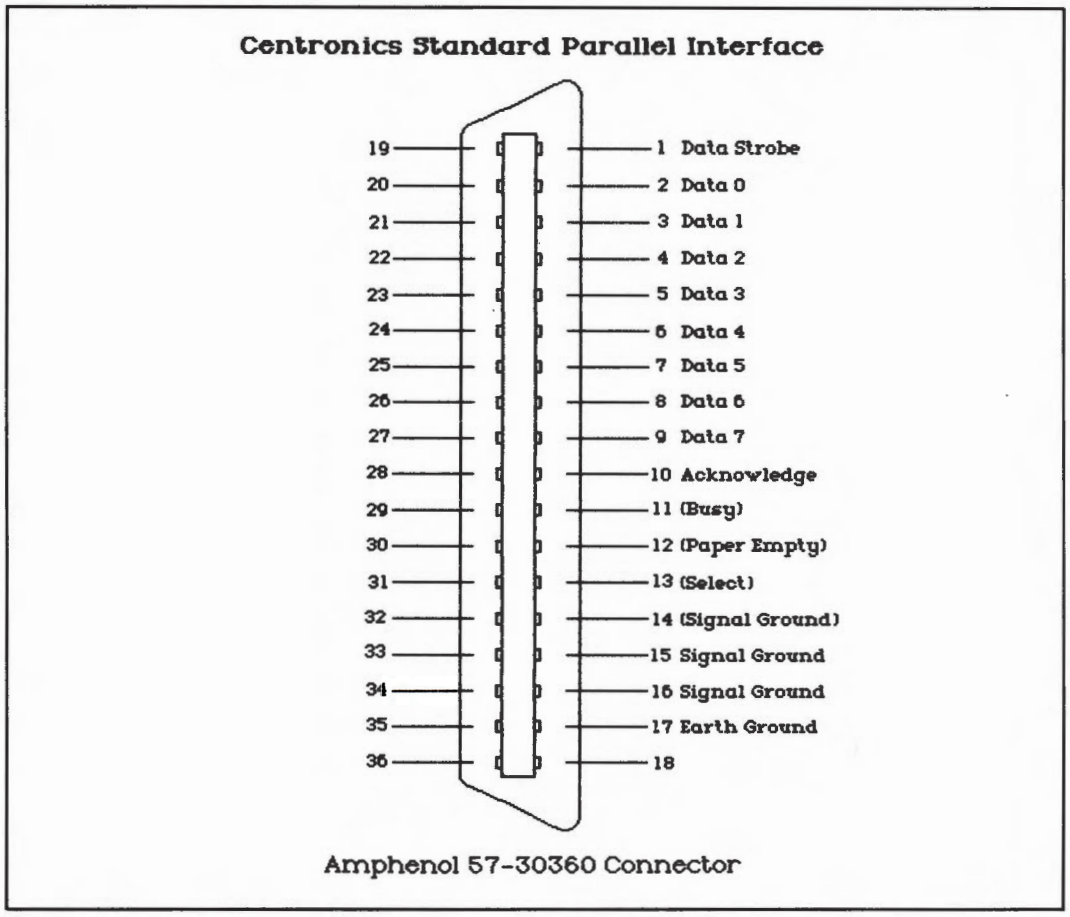


Fig. 18.4 Assignment of Centronics Standard signal lines to pins of Amphenol 57-30360 connector.

For no particular reason the 36-pin Amphenol connector used for the Centronics 779 printer has become the standard connector used for most parallel printers. There is also some variability among manufacturers as to which wire is attached to which pin in these connectors, but the standard system is to put the strobe wire on pin 1, the eight data lines on pins 2 through 9, in ascending order, and the acknowledge wire on pin 10. This pin arrangement is shown in Figure 18.4 and it is usually called "Centronics Standard Parallel."

Once you've gotten compatible connectors and a compatible arrangement of wires and pins in a Centronics style interface, you still have to worry about the polarity of each signal. The most common system has the strobe and acknowledge wires maintained with a steady +5 volt signal, which drops to 0 volts momentarily to signal the actual strobe or acknowledge. This is called a "negative going" polarity. The data lines are usually handled in the reverse fashion, resting normally at 0 volts but popping briefly up to 5 volts whenever a "1" is to be sent.

Parallel interfaces have one more source of incompatiblity even after all connectors, pins, and polarities are settled, and that is the time duration of each pulse. This varies and causes problems even among the most popular printers.

As you can see, interfacing a parallel printer is not any easier than interfacing a serial printer. Therefore, it is just not worth anyone's time to bother with this stuff. If you want to avoid trouble, purchase the most popular printer on the market even if it does not suit your needs exactly, then buy the printer interface card they recommend. One important reason for the success of the Epson MX-80 and the Prowriter dot matrix printers is that they use strictly standard centronics parallel interfaces, and every computer dealer in the country is able to confidently instruct you on how to do the interface.

Another option is to buy a versatile interface card such as the AIO-II card from SSM since it can be made to interface with many different kinds of printers. SSM makes it its business to know the peculiarities of a very large number of printers and will provide you with the appropriate cables and even tell you how to set the switches inside your printer. The downside of buying an AIO-II is that it is so generalized that you must go through a setup procedure.

As long as you realize that you cannot just plug your printer into your interface without checking first with someone, you'll probably be OK. If you do not know any computer owner or dealer who has exactly the same printer and interface card you intend to buy, you should exercise great caution and ask a lot of questions before buying.

Summary of Interface Compatibility

The preceeding sections on the art and magic of printer interfacing can be boiled down to two simple recommendations. First, it does not matter whether your printer has a "serial" or a "parallel" interface. However, it does matter which of several kinds of serial interface or which of several kinds of parallel interface your printer manufacturer has chosen. Second, there is no such thing as a "one size fits all" printer interface card. Unless you know in advance exactly what you need, you should choose a card which can adjust to as many different printers as possible, but still be simple to set up.

The only advantages of a serial interface are that you can use a very long cable (up to 50 feet), and that you may be able to use the one serial interface card to interface with either your printer or an external modem. In the balance, some parallel interface cards give you control of special printing features in dot matrix printers which are very difficult to control with a serial interface. Further, "graphics" parallel interface cards let you use a dot matrix printer to print out "screen dumps" of high res graphics displays. Only the Apple Imagewriter and Scribe printers provide convenient graphics screen dumps with a serial interface.

Among serial interfaces, your best chance is with a printer that controls the flow of characters by using a system called RDY/BSY with the signal being on pin 20 of the RS-232C connector. Parallel printers should use the Centronics Standard with a "negative going" strobe and acknowledge. The terms in this paragraph will be undecipherable unless you plow through quite a bit of stuff earlier in this chapter. Fortunately, you don't have to know what they mean, you just have to be sure that the folks who sell you your printer can guarantee that their machine has one or the other.

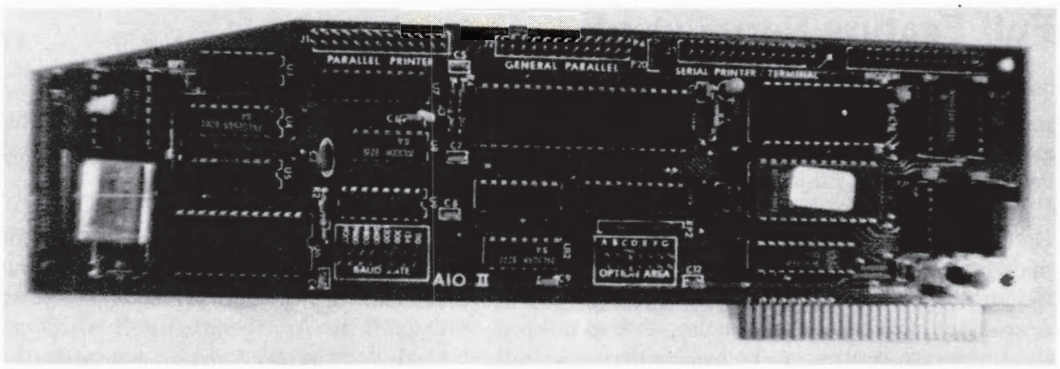


Fig. 18.5 AIO-II parallel/serial interface card from SSM.

If you can't find out how your chosen printer is supposed to be interfaced, you're probably still OK if you choose a very flexible interface card. Any serial printer can be successfully connected to a Super Serial Card (from Apple), an SV622 from Microtek, or an AIO-II card from SSM (see Figure 18.5). Interfacing odd printers with the Super Serial Card is tricky because Apple gives you no advice on how to proceed. The AIO-II is a little better since SSM specializes in knowing how to interface with just about anything and is set up to help you out. The Microtek card is very flexible, but like the Super Serial Card doesn't include information on specific printers.

Printers with odd kinds of parallel interfaces are even easier to deal with. The most versatile interface card is the Microtek Dumpling GX. This card can be set up to handle printers from at least 15 different manufacturers. Better yet, the setup procedure requires that you know nothing other than the name of your printer's manufacturer; no strange codes, no jumpers, no puzzling.

The AIO-II card from SSM can also be interfaced with just about any parallel printer. The AIO-II is therefore the best choice if you are intent on buying a printer from a dealer who can't tell you if the machine uses a parallel or a serial interface. However, setting up the AIO-II for an unusual printer can be a rather involved undertaking. Since the Microtek Dumpling is easy to set up and offers a variety of extra graphics features not available on the AIO-II, it is probably the best choice for use with parallel dot matrix printers.

Serial Interface Cards

Most Apple owners will want to set up their printer card once, close the top of the Apple and never worry about serial interfacing again. Any serial card listed in Appendix G will work just fine, with the exception of the CCS 7710D which only works with modems. The other serial card from CCS, the 7710A, is incompatible with most serial printers making it a poor choice unless you are quite certain it will work with your machine.

If you have any suspicion that your printer has an unusual interface, or if you have any plans to occasionally connect your Apple to some other serial device (a friend's letter quality printer, a modem, a terminal, a plotter), then you should choose a full featured serial card which is easy to convert from one operating mode to another.

Full Feature Serial Cards

The AIO-II card from SSM is the best of the full featured serial cards on the market for several reasons. The first is the commitment of SSM to providing interface information and cables for any printer, terminal or modem which has a serial port. The Apple Super Serial Card is fairly versatile, but you may have a difficult time finding out how to interface it.

Both the Super Serial Card and the AIO-II can be reconfigured either from software or by making hardware changes. The SSC uses DIP switches which are difficult to interpret without a manual, but the AIO-II uses little jumper clips with each feature clearly labeled on the card. A quick glance is always sufficient to find out how you've left the AIO-II configured. Although the hardware setups can be overridden from software, it's very handy to know where you're starting from when things aren't working too well.

If you will be occasionally connecting a modem instead of a printer, the AIO-II contains a built-in "null modem" (see Chapter 16) so that all you need to do is to move the cable to a different position on the card. The Microtek SV-622 also provides both a printer and a modem connector on the card.

Reconfiguring the SSC for modem operation requires the removal and replacement of a DIP "jumper block" and the CCS system requires you to actually buy a second card configured for use with a modem. The AP-SIO from MPC Peripherals uses a jumper block like the SSC, but also permits you to rewire the jumper block to adjust to unusual interfaces.

The Super Serial Card provides a more elaborate set of operation commands than are provided with the AIO-II. These commands will be most useful for programmers who wish to write their own simple routines for listing programs, but since most modern printers provide a dazzling variety of built-in commands, a programmable printer interface card is not as important as it once was.

Operation of the //c Serial Ports

The two built-in serial ports in the //c are based on almost exactly the same chips that are used in the Apple Super Serial Card, the AIO-II, and similar serial interface cards. Each port has its own 6551 ACIA (Asynchronous Communications Interface Adapter) chip, which is described in some detail in Chapter 16.

Although the //c's serial interface hardware is no more elaborate than what you get with a serial card for the II, II+ or //e, the operation of the //c's port system is far more sophisticated. The principal difference between a //c port and a serial interface card is that the //c system relies heavily on "interrupts."

Use of Interrupts in Communications

Interrupts can play an important role in increasing the power of communication software and the printing segment of such applications as word processors. Keep in mind that at a speed of 1200 baud, it takes nearly one hundredth of a second for a single entire character to get shipped out from the 6551 to the serial line along with start and stop bits, parity, etc. During this time, the 6502 could be executing over 1000 instructions.

If the system does not use interrupts, then the 65C02 must spend a considerable amount of its time taking quick peeks at the 6551 to see if it is finished with one character and ready for the next. With interrupts, the 65C02 never has to be responsible for checking with the 6551. Rather, whenever the 6551 requires attention, it can pull the 65C02's "IRQ" line (see Chapter 27).

Configuration Commands for IIc Serial Ports

Printer	Modem	
CTRL-I	CTRL-A	Command Lead-In Character, must precede each command
nnB	nnB	Set baud rate: nn = 1 2 3 4 5 6§ 7 8 Baud = 50 75 110 135 150 300 600q 1200 nn = 9 10 11 12 13 14 15 Baud = 1800 2400 3600 4800 7200 9600 19,200
nD	nD	Set data format: n = 0§ 1 2 3 4q 5 6 7 Data Bits = 8 7 6 5 8 7 6 5 Stop Bits = 1 1 1 1 2 2 2 2
I	I	Echo output to the screen
K	K	§Auto line feed off
L	L	qAuto line feed on (generate and send LF after each CR)
nnnN	nnnN	Set print width/line length to nnn = 0 - 255. (q = 80)
nP	nP	Set Parity: n = 0§q 1 3 5 7 Parity = none odd even mark (1) space (0)
	Q	Quit "Terminal Mode"
R	R	Reset and exit from firmware
S	S	Send 233 millisecond BREAK character
	T	Begin "Terminal Mode," act like dumb terminal, enable RxD and Keyboard interrupts, buffer all serial and keyboard input
Z	Z	§Zap, ignore all command characters and don't format output

q - Default setting for Port 1
§ - Default setting for Port 2

Table 18.3 Configuration commands for //c serial ports.

Using the 6551 to Generate Interrupts

The 6551 is designed to generate interrupts under four different conditions. The first two conditions have to do with sending and receiving individual characters. For transmission, the 65C02 loads a character into the 6551 Transmit Data (TxD) Register and the 6551 goes to work. When it has finished with the character, the 6551 marks a TxD Empty flag in its status register (see Chapter 16, Figure 16.1e) and pulls the IRQ line for attention. There is a similar situation for incoming data, which involves setting a flag for Receive Data Register Full (RxD Full) and then pulling the IRQ line.

The other two interrupt conditions for a 6551 have to do with the state of the external device it is connected to via its RS-232C control lines (see Chapter 17, Figure 17.7 for overview). From the 6551's point of view, there are two conditions under which an external device should cause it to generate an interrupt. The first occurs when the Data Set Ready (DSR) line turns on. This happens when you turn on the power switch of a modem. The 6551 can generate an interrupt at this time to let the 65C02 know that there is a working modem out there. The second situation occurs when a modem has detected an incoming "carrier tone" on the phone line and sends a Data Carrier Detect (DCD) signal to the computer.

Interrupts Available from the //c 6551s

In actual fact in the //c, the 6551s' DSR and DCD inputs are not used for these standard modem connections. In the printer port, the 6551's DCD line usually ends up being connected to the Data Terminal Ready (DTR) output of a printer (see above). Many printers use this DTR line to warn the computer that they have no room for additional characters in their built-in buffer. The 6551 for the modem port is typically hooked up so that its DCD line goes out of the Apple on a pin marked DSR, but which ultimately does get connected to the modem's DCD output.

That accounts for the DCD pin on the Port 1 6551 and for the DCD pin on the Port 2 6551. The DSR pins on the 6551s are used for things that have absolutely nothing to do with the RS-232C system. The DSR pin from the Port 1 6551 is connected to a signal line that runs out through the external disk drive connector and is called EXTINT (external interrupt). This line makes it possible for an external device such as a Z-80 coprocessor module or fancy disk drive system to send an IRQ to the 65C02.

The DSR pin on the Port 2 6551 is connected to the keyboard data strobe line (see Chapter 8). This means that the Apple //c can generate a DSR interrupt everytime you push down a key on the keyboard. When this pathway is activated, a running program can go about doing whatever it wants to, pausing to get new characters from the keyboard only when new data is actually presented.

Handling //c Interrupts

When the IRQ signal arrives at the 65C02, it stops what it is doing and starts scanning through the //c's various interrupt source flags. First it checks the mouse flags in the IOU (see Chapter 9), then the status register (see Chapter 16, Figure 16.1e) in the 6551 for Port 2 (the modem port), and finally the status register in the 6551 for Port 1. There is a total of 11 possible interrupt sources in the //c:

IOU

1. VBL (Vertical Blanking)
2. XO (Mouse X-Move)
3. YO (Mouse Y-Move)

Port 1 6551

4. RxD (Receive Data Register Full)
5. TxD (Transmit Data Register Empty)
6. DCD (Usually the DTR line for Printer Buffer Full)
7. DSR (EXTINT from disk drive connector)

Port 2 6551

8. RxD (Receive Data Register Full)
9. TxD (Transmit Data Register Empty)
10. DCD (Data Carrier Detect from Modem)
11. DSR (KSTRB Keyboard Data Strobe)

Whenever the IRQ signal turns on, the 65C02 can do a quick scan through these 11 flags and figure where the request came from.

Interrupts from the mouse get checked first because VBL gets cleared when its flag is read, and because part of the X and Y position system requires attention within 40 millionths of a second of an interrupt (see Chapter 9). The 65C02 then goes about sorting its way through the 6551 interrupts.

When an interrupt occurs in a //c, there is built in firmware which can "service" some sources automatically. This applies to the mouse interrupts, but it also applies to the two RxD interrupts, the port 2 DCD interrupt, and the KSTRB interrupt. In any of these cases, you can get the //c to respond properly without even loading any software. However, for any of the other interrupts there must be a specially written interrupt handler in your software or else your Apple will just hang.

Terminal Mode

The part of the built-in interrupt handler which deals mostly with serial port 2 is active in the Terminal Mode of the Apple //c. This mode is actually a mini-communications software package built into the //c ROMs. It really doesn't do all that much, but it does showcase the use of the //c's very fancy communications capabilities.

The terminal mode is used to make the Apple //c simulate a dumb terminal. Some writers have likened this to performing a lobotomy on a personal computer. When it is active, no software and virtually none of the firmware will work. The Apple //c is turned into a very modest CRT terminal for some other computer attached via the RS-232C port.

The keyboard can send characters to the other computer, it can "echo" these characters to the screen, and it can display characters sent by the computer. However, nothing even gets stored in RAM, never mind being available for the disk drive. Characters scroll off the top of the screen and they are gone for good. All storage and processing must be done by the remote computer the Apple is connected to. You can, by the way, connect two Apples this way, with one playing the part of the remote computer and the other being the terminal. You type CATALOG on the terminal Apple, but the disk drive turns on in the remote Apple.

Buffers and Interrupts in Terminal Mode

Obviously, this is not intended to showcase the wonderful computational powers of the Apple computer. Rather, it is intended to showcase the way the serial port firmware can handle characters. The most important unique feature is the creation of two RAM buffers in the auxiliary memory in the $0800 page (see Chapter 21). The first 128 bytes of the page (Aux. $0800 to $087F) serve as a keyboard input buffer, while the remaining 128 bytes (Aux. $0880 to $08FF) are the serial character input buffer.

Each buffer has a pointer to where in the page the next incoming character should be stored and a pointer to where the next character should be read from. These pointers are maintained in screen hole locations in main RAM (see Chapter 21, Figure 21.6b) as follows:

$04F7—$C1 for port1, $C2 for port2
$057F—Store next serial character pointer
$05FF—Store next keyboard character pointer
$067F—Read next serial chararcter pointer
$06FF—Read next keyboard character pointer

The key sequence Closed Apple/CTRL-X clears the keyboard character buffer.

In a typical situation, characters from a modem would be flowing into the serial input buffer at a steady rate, with the incoming character pointer advancing steadily for each new character. Meanwhile, the video system firmware would be using the other pointer to read characters for display at an irregular rate (due to scrolling) and it would be collecting them from the keyboard buffer, the serial buffer or both. Thus the buffering system allows for some leeway in the timing of keyboard and serial input vis a vis the video system. However, 128 bytes is not all that much and its certainly easy to overflow the buffers if things get too far out of line.

The input to this system is interrupt driven. Before beginning, the command register of the Port 2 6551 is set up to enable RxD (receive buffer full) and DSR (connected to keyboard data strobe) interrupts. The terminal mode firmware reads the 6551 status register to find out which of the two sources caused the IRQ and responds by grabbing the appropriate character and using one of the pointers to stuff it into the right place in the appropriate Aux $0800 buffer.

Configuration Commands

Before either serial port is used, it must be configured with regards to the asynchronous output format (number of stop bits, data word length, baud rate, parity checking). This information is held in the command and control registers of the 6551 (see Chapter 16, Figure 16.1c and d) when the port is being used, but there must be some simple way of putting the information there and of channging it if necessary. Some serial cards use DIP switches or jumpers to create a default set up format which can be changed from software. The //c default configuration is set up for both ports by the firmware.

When the Apple is first turned on, the firmware copies its default settings into screen holes in the auxiliary memory (see Chapter 21, Figure 21.6b) where they can be altered from software. Later, when a port is activated (i.e., with a PR# or IN#) command, the firmware copies the defaults from auxiliary memory screen holes into main memory screen holes and shoves them into the 6551 registers. Thus, before a port is activated, you can change the defaults in the auxiliary memory, but, after it is activated, you must either stuff the settings into the registers directly, or change the main memory screen hole settings and call the appropriate subroutine.

The easiest way, to configure a port, however, is just to use the CTRL-I and CTRL-A commands as shown in Table 18.3. These commands are similar to those used by the Apple Super Serial Card and other interface cards for the II, II+ and //e, but the command character (CTRL-I or CTRL-A) must preceed each command rather than simply preceeding an entire line of commands.

Simple Parallel Printer Cards versus General Parallel Cards

The simplest type of parallel printer card can support a straightforward Centronics Standard parallel printer and can usually be modified to handle a few other kinds of printers. These cards usually have ROM programs which let you control some detail of program listings such as margins and characters per line. The Apple Parallel Printer Interface (see Figure 18.6) is the only one of these still in wide distribution.

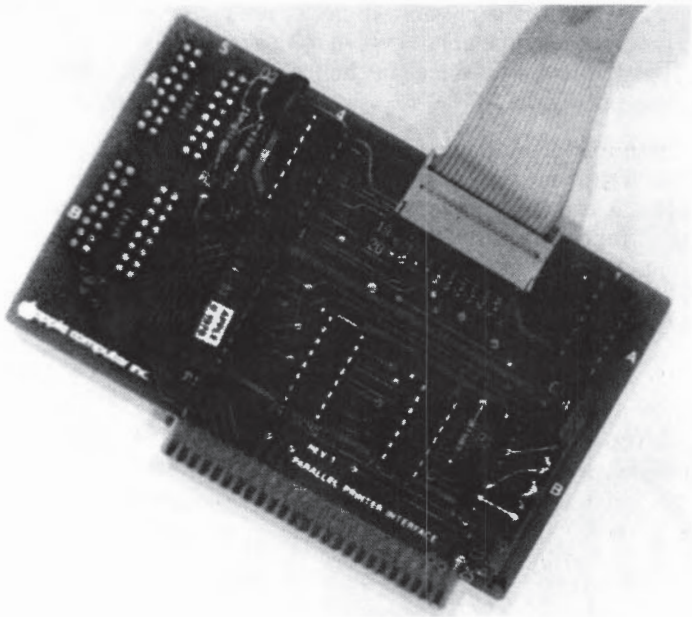


Fig. 18.6 Old Apple parallel printer interface card.

There are some parallel interface cards on the market which are not really designed for printer interfacing. Rather, they are oriented toward the requirements of scientific laboratories or for special effects systems. These cards often use a Motorola 6821 (AIO-II, CCS parallel), or a Synertek 6522 (Fly Board from Snave). The 6821 provides for a variety of special parallel control and data exchange functions and simplifies the task of a programmer with a specialized application. The 6522 is a recent update which adds some timing and simple serial output functions. These kinds of functions are often termed "general parallel."

The Eighth Bit for Controlling Dot Matrix Printers

The most important extra feature for dot matrix printers is the ability of some parallel cards to fiddle around with the eighth bit of every byte. The eighth bit is special because it is never used in standard ASCII character codes. All the ASCII codes can be represented by patterns using only the first seven bits of a byte, therefore, for years, printers and printer interface cards paid no special attention to the eighth bit.

However, as printer manufacturers packed more and more features into their printers, some began including features which depended on the contents of the eighth bit. Many of the best

dot matrix printers now on the market can use eight bit codes. These expanded codes are used to operate some special graphics abilities of the printers and in the newer Epson printers they are used to call up italic characters and special international characters.

There is a problem, however; an Apple must have a special type of printer interface card to use these features. A simple printer interface card may provide no means of using these new printer features.

In fact, it is quite difficult to take advantage of these special eight bit graphics features when one of these printers is connected to an Apple. The problem is that the Apple itself uses the eighth bit for various and sundry internal signaling purposes when it is shuffling codes and characters in a BASIC program. Although it is possible to get around this limitation if you are a skilled machine language programmer, just about everyone else must rely on a hardware fix rather than a software fix. That hardware fix comes in the form of special parallel printer cards with a feature called "eighth bit toggle."

A number of printer interface manufacturers have designed their interface cards to accept commands which can control the eighth bit. These cards take action after the byte has left the Apple motherboard but before it leaves the printer interface card. Simple parallel interface cards with this "eighth bit toggle" function include the SSM APPIC, the RV-611C from Microtek, the Ap-80 from MPC, the Orange Printer Interface (see Figure 18.7), and the Print Max from MicroMax Systems. Parallel cards with an eighth bit toggle typically do not use a 6821 or 6522, but rather rely on a few simple logic gates and direct control from the microprocessor.

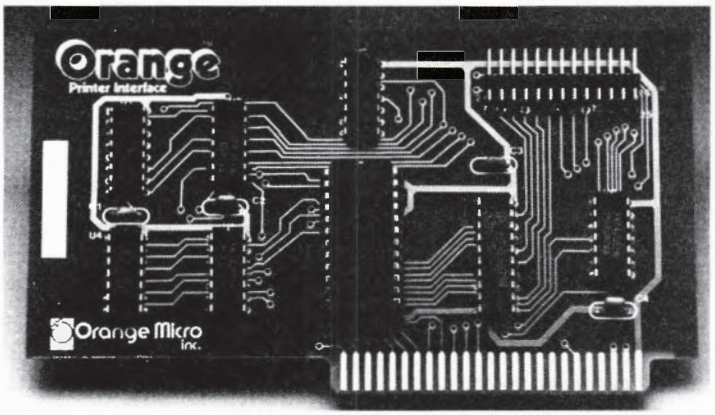


Fig. 18.7 Orange Printer Interface card.

This eighth bit toggle feature lets the BASIC programmer take advantage of a wide variety of graphics commands built into some dot matrix printers (i.e., Epson). Further, it is often possible to trick your word processor into using the eighth bit toggle feature to make some printers type special characters.

"Screen Dumps" for Controlling Printer Graphics

Although the obvious tendency is to think of printers as machines which type just letters and numbers, it is also possible to use many printers to print pictures on paper. In a dot matrix printer, graphics are produced by direct control of the dots in the matrix, and in a fancy letter quality printer they are produced by typing an awful lot of periods and doing amazing things with the print head and the platten.

Many microcomputers have no built-in system for generating video graphics displays, and for these machines, graphics printers provide the only means of drawing graphs, plots or pictures. Apple owners can also use the printer graphics commands to create images if they so desire, but there is a much simpler alternative.

The Apple has a built-in system for drawing graphic images on two high resolution graphics "screens." These images are drawn using commands from Applesoft BASIC, entered with a digitizing drawing board, captured with a video camera, or generated by a wide variety of business graphics and other illustration software.

The idea of a "screen dump" is to let an Apple owner use existing software to draw graphic images on the video screen, but then to use special screen dump software to copy an exact dot for dot replica of the image from the video screen out onto the paper in the printer. A screen dump program examines the binary representation of a video image as it is stored in memory and translates this into a series of commands to the printer's graphics system. You use your own familiar graphics software to generate the images on the screen, and the screen dump software takes care of all the tedium of converting your screen graphic commands into printer graphic commands.

Screen Dump Software and Screen Dump Cards

Screen dump programs come in two forms. In the early days of screen dumps it was necessary to halt the program which had drawn the image, and then to load in the screen dump program from disk. When the screen dump was finished, you then had to reload your regular program if you wanted to draw another image. Screen dump software of this type was written for many dot matrix printers and it was also written for some very fancy letter quality printers.

The next and very important evolution in parallel printer cards was the inclusion of the screen dump software in a ROM chip on the printer card itself. This is extremely important because it makes it possible to call up the screen dump routine from within a running program without any serious disruption or disk drive accesses.

These screen dump graphics printer cards are extremely easy to use. If a BASIC program generates a high res graphics image, the image can be copied to the printer by the use of a simple two line statement:

```
100 PRINT CHR$(4);"PR#1"
110 PRINT CHR$(9);"G"
```

The first line is a simple instruction to turn on the printer card, and the second line is absolutely all you have to write to cause your interface card to make a perfect copy of whatever is being displayed in high res screen 1. (For those who don't read BASIC, line 110 simply sends two ASCII codes to the printer card: CTRL-I and G.) When the printout finishes, your program resumes on the next line with no further fuss.

Graphics Parallel Printer Cards

The first printer card to offer this feature was the Grappler from Orange Micro and this card has become extremely popular. The current Grappler+ offers a variety of additional graphics output features. For instance, if you put in a "G2" instead of a "G" (see line 110 above), you

get a copy of high res screen 2. Other options include the ability to reverse the image to get white on black instead of black on white, to rotate the image 90 degrees on the paper, to double its size or to use emphasized printing to smooth out some of the jagged lines.

Similar graphics parallel cards are manufactured by MBI and by Quadram, and this basic set of features has also been included on several advanced feature graphics cards and special "smart-buffered" printer interfaces described later. The newest version of the Grappler+ can be used with a Revision B Apple //e to print a full super high res image and with the new Epson FX printer to alter the X/Y ratio of images it is printing.

The Grappler Interrupt Problem

Unfortunately, the Grappler has one serious design flaw and other manufacturers are now offering cards with a wider variety of graphics dump features and without the flaw, the best of these being the Dumpling GX from Microtek. The flaw is potentially very devastating if you are using DOS rather than ProDOS. The newer Orange Printer Interface and Buffered Grappler+ show a redesign to correct the problem, but the Grappler+ is still being sold without modification.

The problem is that although the 6502 and the Apple seem as though they ought to be able to handle "interrupts," there is a bug in the design of the Apple's firmware which makes it very dangerous to actually use them. Apple has been quite forthcoming about this bug (which is described in Chapter 27) and has always warned peripheral manufacturers not to use interrupts.

Some printer cards have interrupt circuitry for use in the Apple III, but all of these cards are shipped with the interrupt line electrically disconnected for use in Apple II, II+ or //e computers. The use of interrupts can cause intermittent and bizarre effects including the occasional destruction of disk files. ProDOS can handle interrupts, but must be specially modified to do so properly with the Grappler+. Apple //e's sold after the summer of 1984 have a revised set of Monitor and I/O ROMs which handle interrupts properly. These ROMs are available as an upgrade for any //e.

The Grappler+ uses interrupts to warn you if your printer isn't ready to go. It stops everything to flash a message on the screen telling you that your printer isn't working (which you presumably already know because you don't hear it clacking away). Depending on exactly what was going on in the computer when the interrupt was generated, this can have very unpleasant consequences. If you are using a Grappler+ with DOS you should be very careful to be sure your printer is on line and ready to go whenever the Grappler tries to call it. The Microtek Dumpling GX is an excellent alternative. Although the Buffered Grappler+ has been redesigned to remove the interrupt circuitry altogether, it has other problems described below.

Advanced Feature Graphics Printer Cards

The kinds of additional features appearing on other cards include screen dumps of selected lines on any one of the text screens, text screen dump of lowercase letters in the //e, automatic formatting of screen dumps of BASIC program listings (starting a new line at every colon), and "transparent eight bit data transmission" where the toggle system is disabled (for simple transmission of binary data); all of which are provided by the Dumpling GX.

The Pkaso board from Prometheus can interpret lo res color data to vary gray scales on its printouts, and both the Macrotech Graphics Parallel card and the GraphMax from MicroMax add the ability to zoom in on any part of an image over 16 levels of magnification.

The most innovative and useful of all is the Print-It card from Tex Print (see Figure 18.8). With this card installed you are able to press a button at any time during any program including, for instance, Apple Writer on your Videx 80 column card, and get a complete image copy of whatever is on the text screen. The board is based on the Apple Non-Maskable Interrupt (NMI) line (see Chapter 27) which is used in some copy cards. It halts the processor, copies the appropriate screen, and then returns you to whatever you were doing just before pressing the button. In addition, the card can be operated in serial mode for screen dumps to Apple's serial Imagewriter.

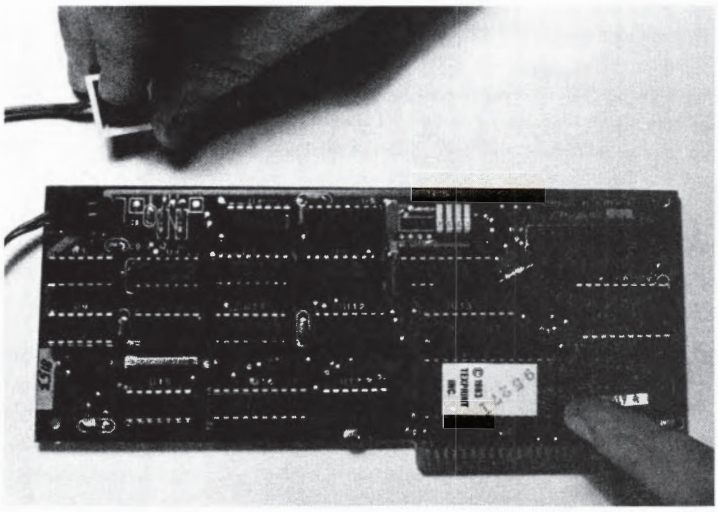


Fig. 18.8 Print-It screen grab printer interface from TexPrint. The button uses the Apple's NMI system to halt the computer and copy text or graphics from the screen to the printer.

Multi-Function Interface Boards

For those fortunate Apple owners who have all of a dot matrix printer for fast text printouts and graphics, a modem for communications, and a letter quality printer for formal documents, a different sort of interfacing problem arises. Where do you put all those interface cards? One solution is to attach all of them to the one card which has several ports. This may cut down on the total you have to spend on interface cards, reduce the demands on your Apple's power supply, and leave more slots open for other cards. There are a few hitches, however.

In theory, the Apple is only able to speak to one device in each slot. Designers of multi-function cards have chosen two routes to get around this Apple singlemindedness. The simpler of the two approaches is used by SSM in its AIO-II board and by U-Micro in its multiple port serial board. In these cards, several devices can be connected, but the Apple is aware of only one of them at any given instant. If the card is in slot 2, then all of the devices attached to the card are addressed as if they were really just one single occupant of slot 2.

A BASIC or machine language programmer can give commands to switch the output among eight different serial ports on the U-Micro card. The AIO-II is a little more clever in that it permits you to send each byte simultaneously to both a serial and a parallel port. The Apple

is only aware of the serial port, but the parallel port gets a duplicate copy of everything passing through the card. In this way, modem communications through the serial port can be "echoed" to a parallel printer.

A more sophisticated approach is used in multifunction cards from Mountain Computer (the CPS), from Videx (the PSIO), and from Prometheus (the Versacard). In these cards, some rather subtle trickery is played on the Apple so that it thinks it is talking to several different cards placed in several different slots, when in reality there is only one card present. The details of this trick are explained in Chapter 22. However, it is a fairly simple trick and works quite well.

The Videx PSIO board can be set up to look as if it has a parallel printer port in slot 1 and a serial modem port in slot 2. The Mountain CPS card can do the same thing, but it also includes a real time clock/calendar which can be "phantomed" into slot 7. The most elaborate of the multi-function boards is the Versacard from Prometheus. It provides a screen dump graphics parallel port appearing in slot 1, a serial modem port for slot 2, a "BSR" power line controller (to turn on/off various switches and controls around your house; see Chapter 14) in slot 4, and a clock in slot 7 (these slot assignments are just examples and can be changed to suit your system).

Printer Buffers and Intelligent Interface Cards

One annoying aspect of using a printer to type long documents is that printers need constant attention from a computer in order to keep working. They don't require the computer to do anything very fancy, but they uncompromisingly demand its steady, plodding and unbroken attention. For most folks, this means that while their printer is working, the Apple is also tied up and you can't get any other work done until the printer is finished. If you saved money by buying a slow printer, this can mean sitting around for hours watching your Apple work without being able to use it.

The good news is that since the printer's requirements are fairly unsophisticated, it is possible to build a simple, inexpensive little "micro-micro" computer to supervise your printer, and a number of manufacturers have done so. These devices usually have some sort of minimal microprocessor, a sizeable amount of RAM memory, some control program stored in ROM, and two interface ports in any mix of serial or parallel. Such printer supervising devices go by the rather unassuming name of "Printer Buffers."

The usual protocol is for the Apple to set up a high speed conversation with the printer buffer's microprocessor, and to hand across everything to be printed in a matter of seconds. The microprocessor in the printer buffer stores everything in its own RAM memory and then goes about the slow plodding process of spoon feeding the characters to the printer. For instance, a 50 page document may take 90 minutes to print on a slow letter quality printer, but the Apple will finish up all of its responsibilities in just a few minutes.

This is also very handy in graphics screen dumps from within a program. If you are drawing graphs and charts on the video screen, the screen dump can be made to a printer buffer in a few seconds. You go on to setting up the next graph almost immediately while the printer plods along copying the image onto paper under supervision from the printer buffer's microprocessor.

Sizes and Shapes of Printer Buffers

There is an enormous range of abilities among printer buffers, from very simple devices that capture and transmit small 16K files to powerful stand alone devices with four times as much memory as the Apple and capabilities for very sophisticated control of multiple documents and graphics. The smartest and most advanced, the Shuffle Buffer from Interactive Structures (see Figure 18.2) provides the only means for an Apple owner to conveniently insert charts and displays within the body of text from a standard word processor.

The three major varieties of printer buffers are: one, Apple plug in cards which include both the buffer system and the complete guts of a printer interface card; two, plug-in cards which have only a buffer system; and three, external or "stand alone" buffers which are sort of inserted into the cable between your printer interface card and your printer.

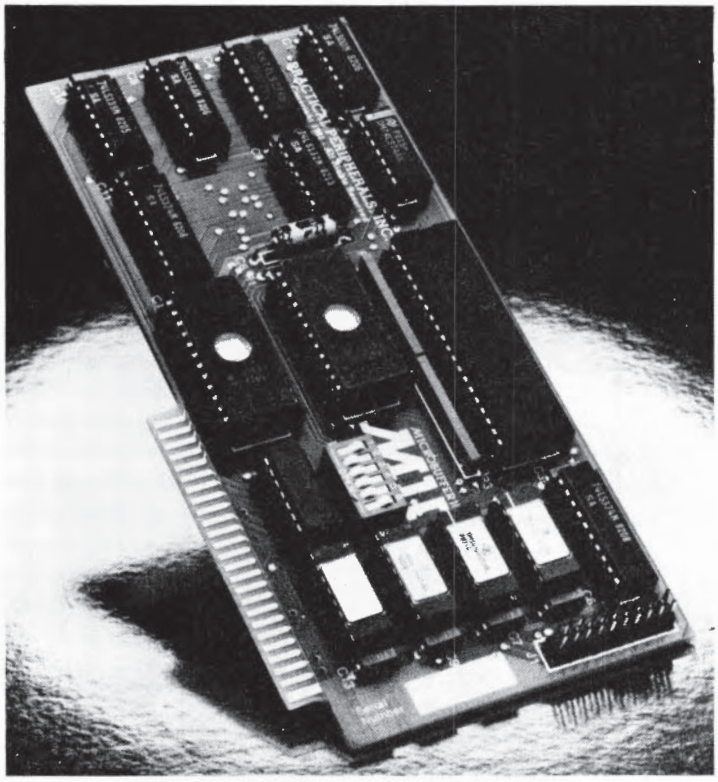


Fig. 18.9 Microbuffer II from Practical Peripherals.

Buffered Printer Interface Cards and Buffer Boards

The Dumpling 64 from Microtek is the most elaborate buffered printer interface board. It is similar to the Buffered Grappler from Orange Micro and the Microbuffer II+ ($349) from Practical Peripherals, and like these other boards it has a maximum capacity of 64K for stored documents. All of these cards provide a full range of graphic screen dump commands for parallel printers and make it completely unnecessary to buy any other printer interface card. The Microbuffer II+ (see Figure 18.9) is also offered as a serial interface without graphics capabilities.

These three cards are, however, very different devices. The problem is that during a long printout you may want to interrupt your printer momentarily to get a little temporary room silence or to zip out a quick one page letter. The only way to stop the Buffered Grappler is to hit reset or turn off the Apple; this may mean you will have to start over from the beginning. The Microbuffer is only a little better since a CTRL-Z sent to it will cause it to stop, but will also clear the buffer's memory.

The Dumpling 64 is much more sophisticated. With this card, you can make the printing pause at any time, bypass the buffer to send data directly to the printer, then resume printing exactly where you left off. Several documents can be loaded into its memory along with instructions to pause between documents for further instructions, and it can even be connected to your telephone and be made to pause automatically whenever the phone is picked up.

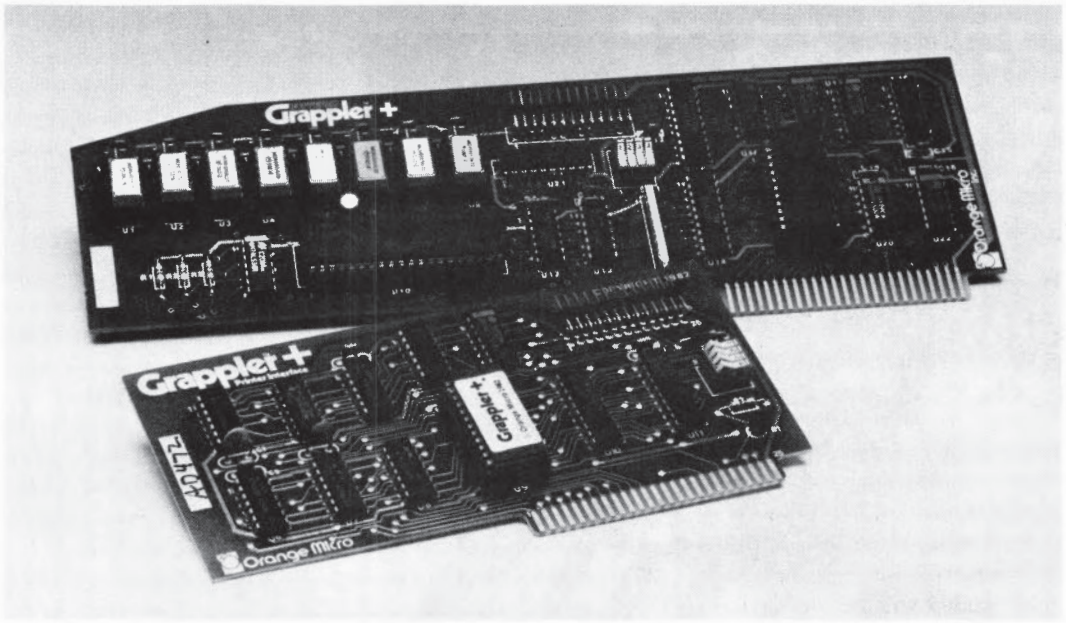


Fig. 18.10 Grappler+ (bottom) and Buffered Grappler+ (top) from Orange Micro.

The Dumpling 64 also offers an important additional feature called Space Compression which allows documents up to about 90K in length to be packed into its 64K of memory, and may also permit up to 200K of graphics images to be squeezed in. Compression is based on counting up strings of similar bit patterns and replacing them with a brief repeat instruction. In text, for example, spaces can be replaced with tabs. In graphics, whole regions of light and dark can be represented by very brief abbreviations. Since the Buffered Grappler (see Figure 18.10) from Orange Micro offers only the Reset button and fewer graphics commands, the Dumpling 64 seems to be far and away the optimum choice for a buffered graphics parallel card.

If you already own a printer interface card you can purchase a card for an Apple slot which provides just the buffering ability. These cards are sold by Orange Micro (the Bufferboard; see Figure 18.11) and by Prometheus. Like the Buffered Grappler, the entry from Orange Micro in this field can only be stopped with the Apple's Reset key. The Prometheus Versabuffer offers both a serial and a parallel port on the same card so you can feed data into it from a parallel printer interface card, but interface with either a parallel or a serial printer.

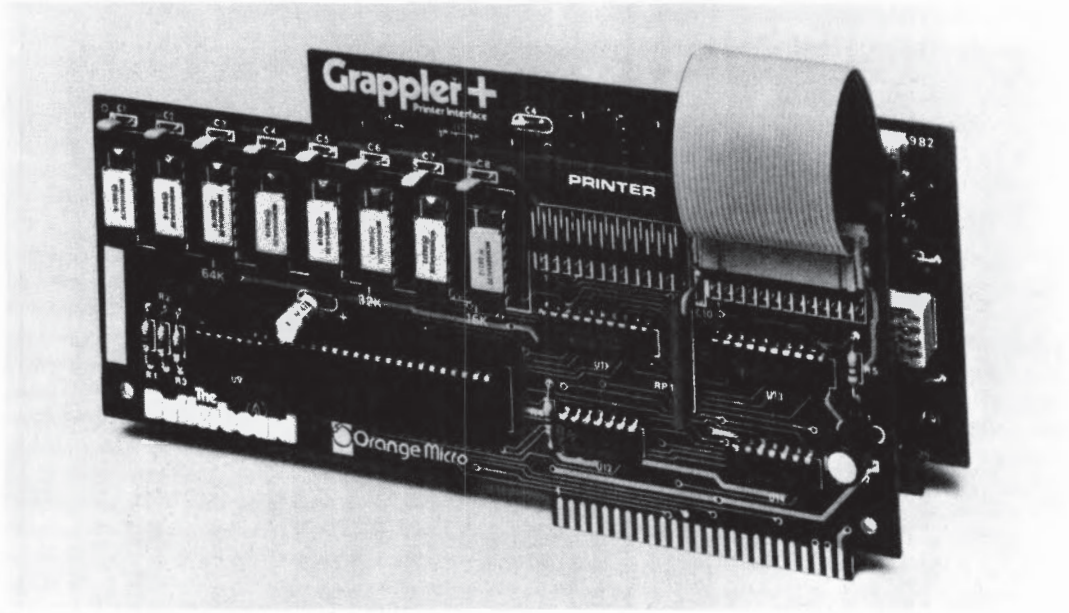


Fig. 18.11 Grappler+ with bufferboard.

External Printer Buffers for High Capacity and Extra Features

Smart buffer systems that are placed on the table next to the Apple are made by MPC (the Omnibuffer; see Figure 18.12), by Practical Peripherals (the Microbuffer; see Figure 18.12), by Quadram (the Microfazer, and the Interfazer), by Microtek (the Buffer Box), and by Interactive Structures (the Shuffle Buffer which updates the Pipeline; see Figure 18.2). The Microfazer ($299) and the Omnibuffer are limited to 64K, but the Omnibuffer accepts either serial or parallel input and connects to either parallel or serial printers. The Shuffle Buffer and the Buffer Box can hold up to 128K of RAM, while the Microbuffer and the Interfazer can be loaded with up to 256K of RAM chips.

Despite varying RAM capacity, all of these external buffers provide convenient pause features and also can be set to generate multiple copies of whatever document is in their memory. The serial version of the Microbuffer is unique in that it can handle bidirectional information flow. This means it can capture data coming into it from a modem as well as send data to a printer. The Microtek Buffer Box offers the data compression and remote pause features that were described earlier for the Microtek Dumping 64 card, and the Quadram Interfazer can be expanded to interface with several different computers and several different printers.

The Shuffle Buffer from Interactive Structures is much more sophisticated than the other external buffers and is the only one which can really be called a "smart" buffer. This buffer can be loaded with up to 63 different files, containing either text or graphics, but its truly unique feature is that the various files can be printed out in any order the operator selects. For instance, you could break up a long text into several short files, and then load it into the Shuffle Buffer followed by 10 or 15 graphs and charts. The Shuffle Buffer is then given instructions on the order in which to print the files. You could print some text, then have the buffer insert a chart, then more text, etc. The order in which they are printed is independent of the order in which they are fed into the printer, and can be supervised by a special command file also loaded into the buffer for unattended operation.

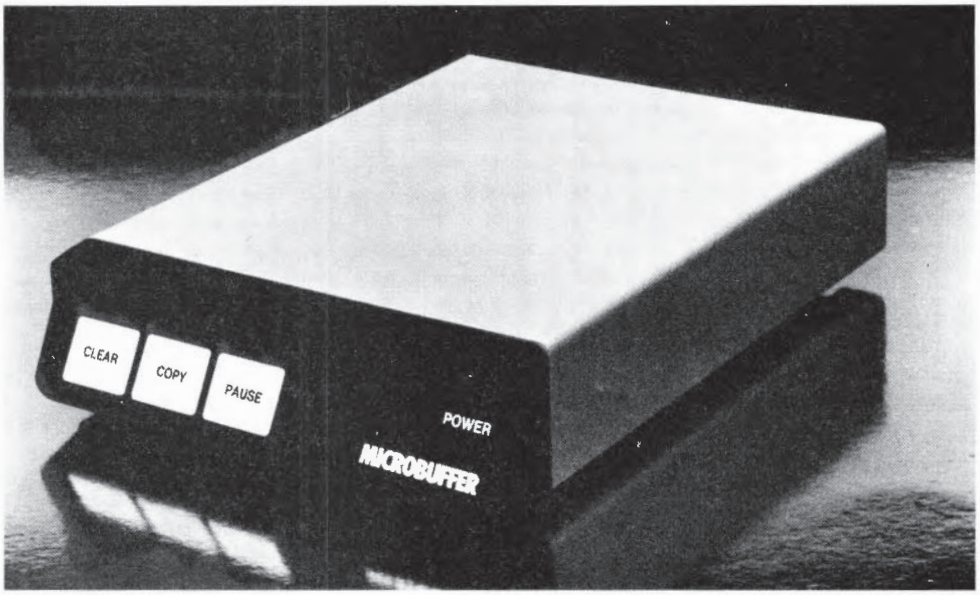
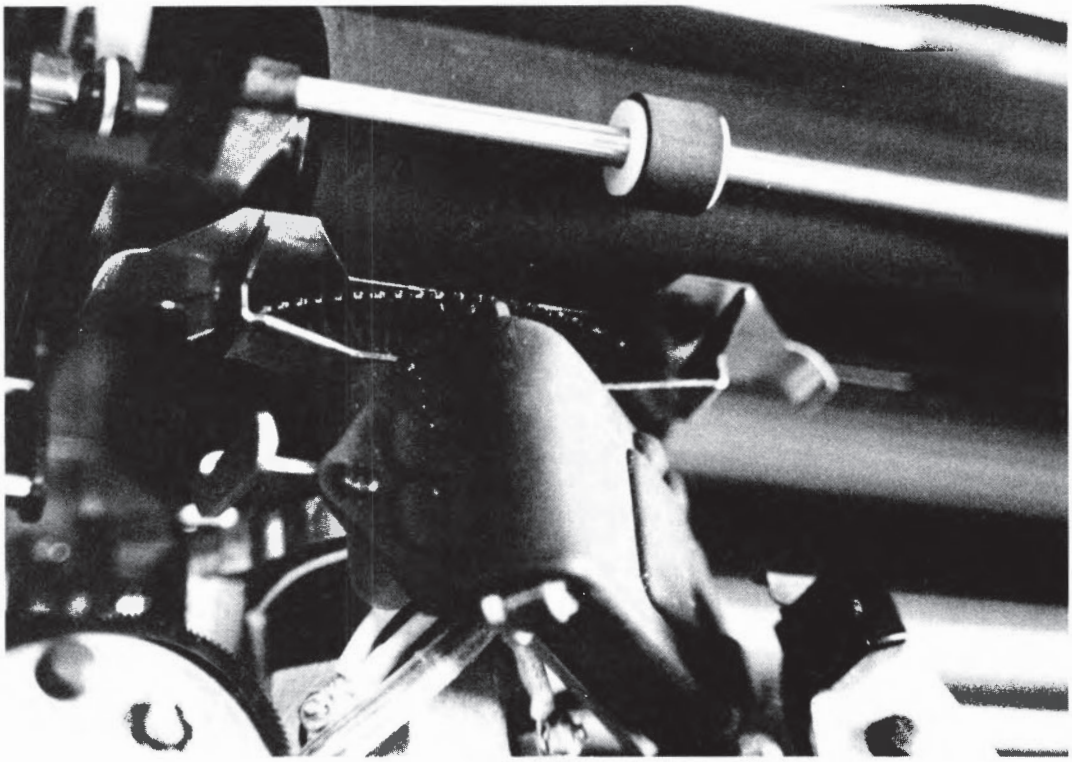


Fig. 18.12a Microbuffer In-Line from Practical Peripherals.

Fig. 18.12b Omnibuffer from MPC.

Fig. 18.12 External printer buffers.



Chapter 19

Printers and Plotters

Printers are Not Like Typewriters

Some Apple owners who buy printers get badly burned by the experience because of a little confusion about what kind of a beast a printer really is. The obvious analogy is to a typewriter, so printer manufacturers happily snare potential buyers by advertising typing speed and the appearance of characters on paper. These two features are important, but they are also quite misleading. A letter quality printer such as the Qume Sprint 11 or Starwriter F-10 is more like a very high strung typesetting machine than like a desktop typewriter.

Control Systems for Letter Quality Printers

A good word processing program, such as WordStar, working with the Starwriter has precise control over the position of each letter on the page down to the level of less than 1/150th of an inch. Strength of impact is varied for each character so all inking is equally dark, and letters are positioned via a trick known best to calligraphers where thin letters are allotted less space on a line than wide letters. There are in fact dozens of tiny subtle formatting instructions that the Starwriter will respond to in the production of elegantly arranged printouts.

This all may sound very complex, but the beauty of the system is that all of this is done without any awareness on your part. The bad news is that virtually each printer on the market has its own coding system for the formatting controls. You may pay nearly $2000 for one of these marvelous machines and never have even the slightest prospect of using it as anything other than a crude but fast typewriter (or "teletype-like" printer) all because your word processing program doesn't know the codes for your printer. Worse still, many very well known letter quality printers lack two or three crucial formatting commands which renders the whole system inoperative; once again leaving you with a very expensive typewriter.

Dazzling Abilities from Dot Matrix Printers

Buyers of dot matrix printers often fall into an even worse version of the same trap. The tendency of many buyers is to think of a dot matrix printer as a cheap alternative to a letter quality printer with poorer looking characters but greater speed. Not so. The Epson FX-80 (see Figure 19.1), as shipped standard from the factory, can type in any one of over 100 different type styles. It responds to more than 80 commands and gives the graphics programmer individual addressable control of over five million dot positions on each 8 1/2 by 11 sheet of paper.

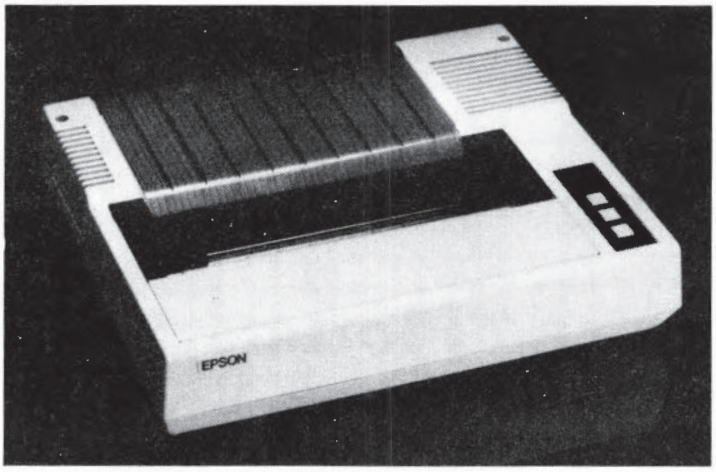


Fig. 19.1 Epson FX-80 dot matrix printer.

Through crafty use of these innumerable commands, an Epson FX-80 or the slightly less elaborate ProWriter 8510 can be made to simulate the typesetting abilities of a very fancy letter quality printer. It is for instance possible to prepare a version of WordStar which operates an Epson or a ProWriter so that the positions of the characters exactly mimic what would be produced by a Starwriter including superscripting, subscripting, and spacing changes, but in addition using shifts in type font for italics and various international or mathematical characters.

Compatibility and Documentation

The bad news once again is complete incompatibility of commands among the various dot matrix printers. Software is readily available for effortless use of many of the features of the Epson printer series, and for some of the features of the ProWriter, but other printers may be reduced to operation as rough quality typewriters, and all the money you pay for the extra abilities will be wasted.

The Apple Imagewriter (see Figure 19.2) is based on the mechanical parts of the ProWriter, but the serial electronics, escape sequences and case have been redesigned to make the machine run faster, do more tricks, and work more quietly ("53 dB" is the figure of merit you can use for comparisons). The Imagewriter presents yet another set of unique escape codes built around the needs of the bit graphics system of the Lisa and MacIntosh computers. You can't expect Epson or ProWriter software to fully utilize this machine, but the weight of Apple's support may make this a safe way to go.

If you are interested in learning to use the various special features on your own, as you learn to program in BASIC, then absolutely buy an Epson and no other dot matrix printer. This caveat to go with the biggest manufacturer may offend some people but there is a good reason, and that reason is a book by David A. Lien titled *The Epson MX Printer Manual* which comes with your printer. It is both a valuable BASIC tutorial (although a little rough around the edges on Applesoft itself) and an excellent way to learn to control a dot matrix printer.

The manual distributed with the newer Epson FX-80 is not quite so entertaining, but it is 300 pages of clear writing. You must understand that the manual for the ProWriter covers the same ground in about 60 pages and can be deciphered only by machine language programmers. Many other printers just don't tell you how to use the features, and some others don't even have the features.

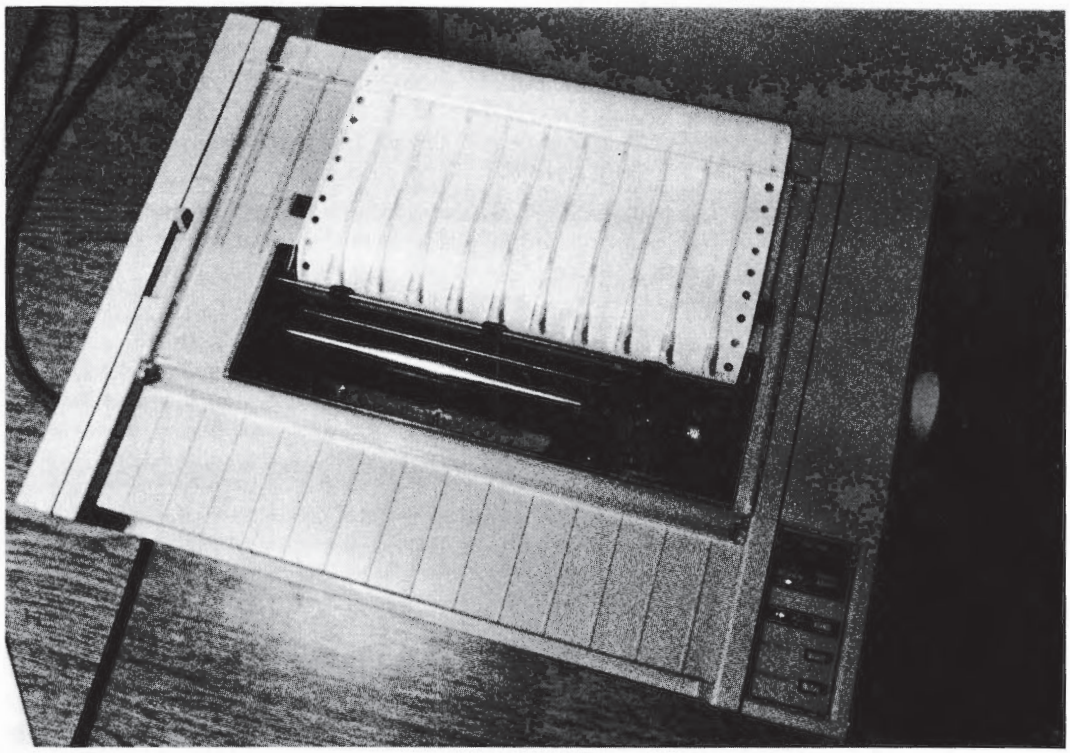


Fig. 19.2 Apple Imagewriter dot matrix printer.

Selecting a Printer

There are over 100 dot matrix and letter quality printers on the market, and you can spend weeks reading reviews that discuss character formation, speed and price. But all this misses the point. Only a small handful of these printers are actually compatible with word processors such as WordStar. Of these, not all are rugged and reliable. Worse, if your ribbon runs out at 3 in the afternoon and you have to submit an important document by 5, you are absolutely out of luck if your local computer store doesn't carry your printer's kind of ribbon. Some printers end up being shut down for weeks waiting for a new ribbon to arrive from a manufacturer. Really, this actually goes on.

Dot Matrix

If you want a reliable dot matrix printer at a good price which works hard and jumps through hoops without any fuss, buy an Epson or a ProWriter (Imagewriter). There are less expensive "compatible" printers which may handle the software, but be sure your dealer can handle service and support.

These printers use a print head which has nine hammer wires stacked one above the other and which are struck home seven (MX-80, ProWriter) or nine (RX-80, FX-80) times in rapid succession to form each character (see Chapter 2, Figure 2.2). Epsons and ProWriters are available with fully standard versions of the parallel or serial interfaces, while the Imagewriter is available only as a serial printer.

The ProWriter is the least expensive ($495), types at about 120 characters per second, and does fairly good graphics, but the documentation is very poor. The Epson MX-80 ($525) is the universal standard among dot matrix printers. It lacks some of the versatility of the ProWriter, and only types at 80 characters per second, but it is the best documented printer for the BASIC programmer. The souped up Imagewriter from Apple can do 180 characters per second. Its documentation is better than for the ProWriter, but not as good as the Epson.

Of the two new printers from Epson, the RX-80 ($495) adds some graphics capabilities and runs a bit faster than the MX and includes a built-in tractor feeder for standard paper.

The FX-80 is a marvelously versatile machine and there is really no other printer on the market with this much speed (160 characters per second), this many type fonts (128 shipped with the printer), this density of dot graphics (240 dots per inch), all together with compatibility with most dot matrix software on the market, reverse paper feed and backspace capability, and a list price of $699. All of these printers are available in wide carriage models (see Figure 19.3) which cost a bit more.

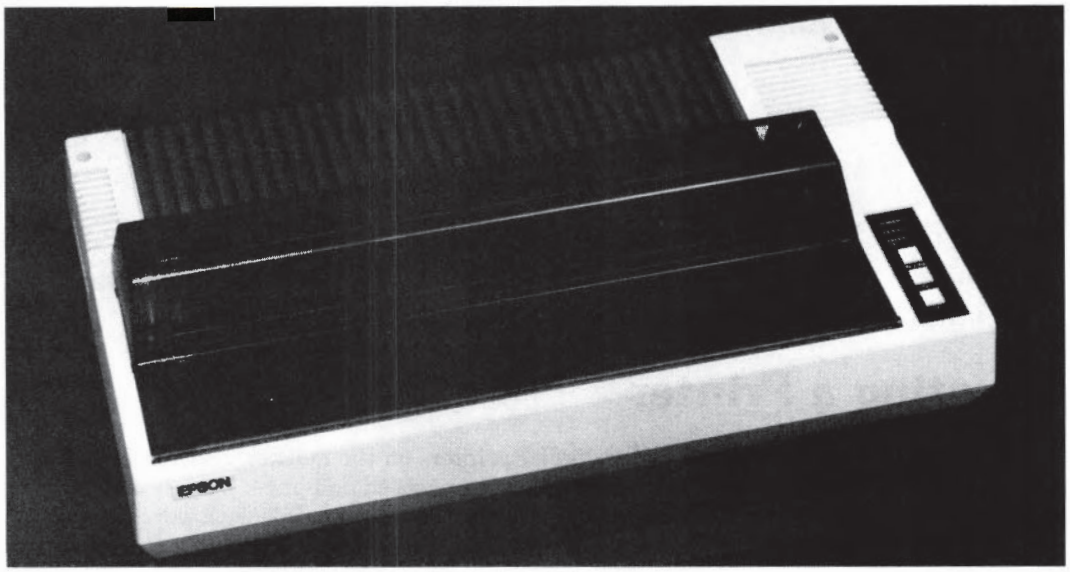


Fig. 19.3 Epson FX-100 for wide paper.

You should note that the "characters per second" figure given for a printer doesn't really tell the whole speed story. In the literature from most manufacturers, this only describes the speed at which the print head advanced along a single line of solid text. The speed at which the head changes direction, speed of platten advance, and even form feed time can cut the "characters per second" speed in half when measured for an entire document.

The Toshiba P1350 is the first of what will no doubt be a new generation of dot matrix printers. It is the first dot matrix printer to really compete directly with daisy wheel printers for print quality and formatting control. Most dot matrix printers have nine pins stacked one above each other, so there will inevitably be spaces between the dots. The Toshiba P1350 has a staggered pin arrangement so there are no blank spaces, the head of each pin is about one half the diameter of the Epson pins, and the staggered array has 24 pins instead of nine. Characters can be formed in a 24 by 18 matrix instead of the Epson FX-80's nine by 11.

This high density head produces characters which are very hard to distinguish from daisy wheel print, but which can be typed at 100 characters per second. One major technological obstacle to this sort of print head has been the problem of keeping such fine wires from getting

caught in the ribbon, and so Epson will soon be releasing a similar print head in which the ink is applied to the pins without use of a ribbon.

Toshiba has also included a unique software control system. The P1350 is able to emulate a Qume Sprint Five daisy wheel printer. This means that a word processing program such as WordStar can be told to send "escape codes" which normally take care of proportional microspacing, superscripting, subscripting etc., for a Qume printer, and the Toshiba will execute those commands.

If you are not concerned about software compatibility, are willing to be a bit of a pioneer, and cannot tolerate printer noise, then there is an important new alternative. Siemens is now selling a printer that acts like a high speed dot matrix machine, but uses a stack of nine ink jets instead of nine wires. The ink sprays quietly onto the paper and there is no hammer. The PT-88 is more than twice as quiet as an Epson even at 150 characters per second. A high speed model, the 2712, can print almost as quietly at nearly 270 characters per second but costs over $3000.

Letter Quality Printers

You will probably get very good performance out of letter quality printers from TEC, NEC, Qume and Diablo, but, as you can see from Table 19.1, their escape codes vary from model to model even for a single manufacturer. The Letter Quality Printer from Apple Computer is

	Diablo 630	NEC 7710 7720	NEC 7715 7725	QUME Sprint 5	TEC F-10 Star-writer
Print Head Control					
Set Up					
left margin set	ESC 9	ESC m	ESC 9	ESC 9	ESC 9
" clear		ESC O			
right margin set	ESC 0	ESC J	ESC 0	ESC 0	
" clear		ESC K			
horiz. tab set	ESC 1	ESC 1	ESC 1	ESC 1	ESC 1
" clear	ESC 8	ESC 2	ESC 8	ESC 8	ESC 8
list of horiz.tabs set				ESC (	ESC (
" clear				ESC)	ESC)
clear all tabs	ESC 2			ESC 2	ESC 2
set horiz.increment	ESC nn_	ESC l	ESC nn_	ESC nn_	ESC nn_
" clear	ESC S				
Motion Commands					
carriage return	m	m	m	m	m
horizontal tab	I	I	I	I	I
absolute horiz.tab	ESC nn I	ESC	ESC nn I	ESC nn I	ESC nn I
relative horiz.tab				ESC H	ESC H
back space	H	H	H	H	H
Word Processing Format Controls					
underline on	ESC E	ESC -	ESC E		
" off	ESC R		ESC R		ESC R
bold print on	ESC O	ESC *	ESC O		ESC O
" off	ESC &		ESC &		ESC &
shadow on	ESC W	ESC +	ESC W		ESC W
" off	ESC &		ESC &		ESC &
bidirectional on		ESC)	ESC /		
" off		ESC (	ESC /		
auto centering	ESC =	ESC $A	ESC =		
" off		ESC &			
auto justify on	ESC m	ESC $B	ESC m		
" off		ESC &			
forward printing	ESC 5	ESC >	ESC 5	ESC 5	ESC 5
reverse printing	ESC 6	ESC <	ESC 6	ESC 6	ESC 6
proportional spc. on	ESC S	ESC H	ESC P		ESC I
" off	ESC Q	ESC I	ESC Q		ESC J

	Diablo 630	NEC 7710 7720	NEC 7715 7725	QUME Sprint 5	TEC F-10 Star-writer
Paper Movements					
Set Up					
top margin set	ESC T				
- clear	ESC C				
bottom margin set	ESC L				
- clear	ESC C				
set lines per page	ESC nn L	ESC L	ESC nn L	ESC F	ESC F
vertical tab set	ESC -	ESC 5	ESC -		
- clear		ESC 6			
set vert. increment	ESC nn ^	ESC]	ESC nn ^	ESC nn ^	ESC nn ^
Motion Commands					
line feed	J	J	J	J	J
reverse line feed	ESC nn J	ESC 9	ESC nn J	ESC nn J	ESC nn J
half line feed	ESC U	ESC	ESC U	ESC U	ESC U
negative half line feed	ESC D	ESC	ESC D	ESC D	ESC D
form feed	L	L	L	L	L
absolute vertical tab	ESC nn K		ESC nn K	ESC nn K	ESC nn K
relative vertical tab		ESC X		ESC V	ESC V
Print Wheel Control					
select lang./wheel size	ESC nn V				
program mode on	ESC nn n	ESC E	ESC nn n	ESC nn _	ESC nn n
- off		ESC G	ESC nn		ESC nn I
print spec.char.#1	ESC Y			ESC SP	ESC SP
print spec.char.#2	ESC Z			ESC /	ESC /
graphics mode on	ESC 3	ESC /	ESC 3	ESC 3	ESC 3
- off	ESC 4	ESC	ESC 4	ESC 4	ESC 4
Ribbon Control					
black	ESC B	ESC 4	ESC B	ESC B	ESC B
red	ESC A	ESC 3	ESC A	ESC A	ESC A

Table 19.1 Escape and Control codes for letter quality printers.

based on a Qume. The Starwriter from TEC (C. Itoh, Leading Edge) offers good performance at a lower price though its codes are close to, but not the same as, Qume's codes. The best approach may be to first choose a word processing program and then select a printer only if the software company says their program can operate it.

These printers are called "letter quality" because they use fully formed letters on a daisywheel (see Chapter 2, Figure 2.3) or thimble to print their characters, but they really earn this name because of their powerful and professional control over character positioning. Most can be

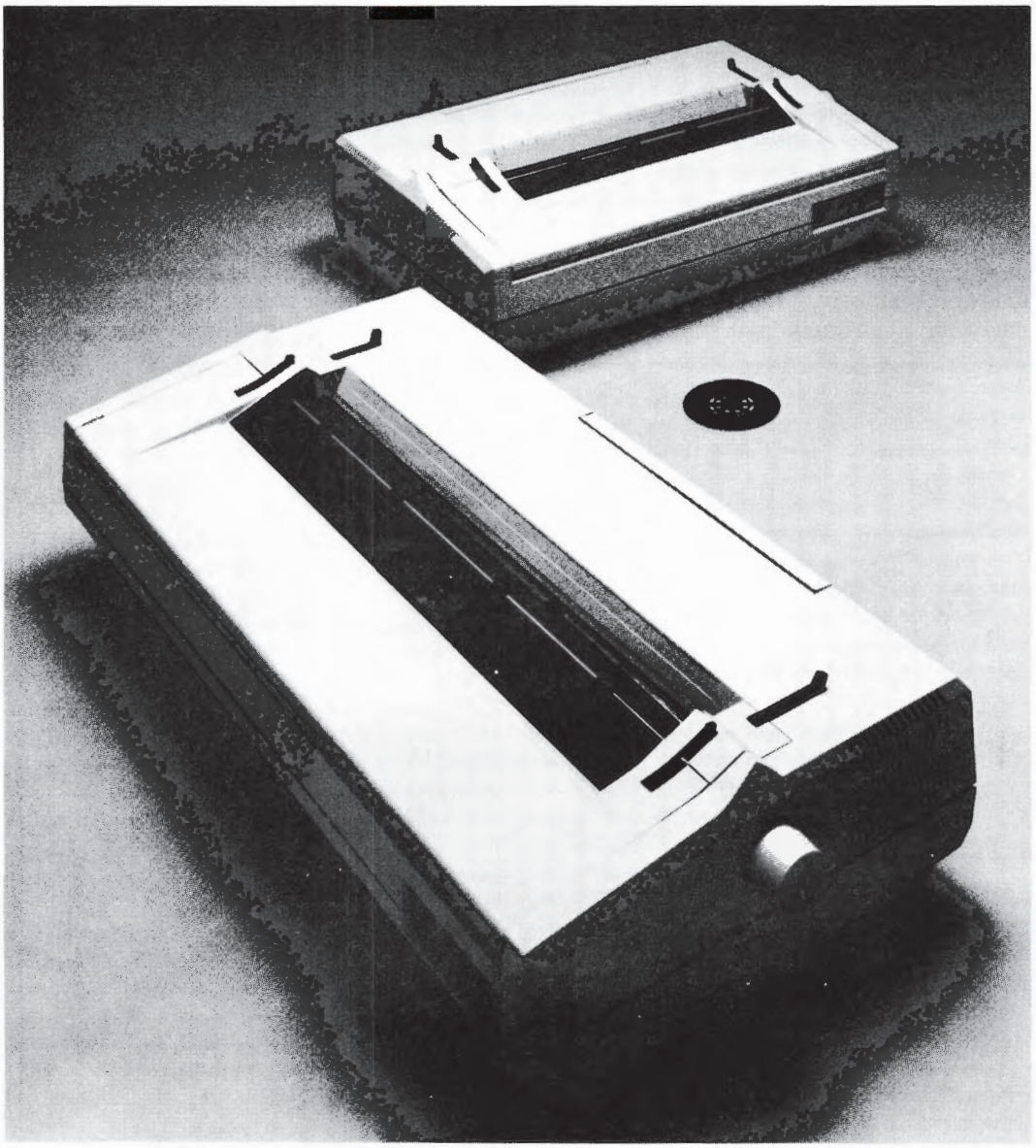


Fig. 19.4 Qume Sprint 11/40 widetrack (front) and standard 15 inch (back). Available with 130-character daisy wheel or standard 96 character wheel (40 or 55 characters per second).

programmed to use a wide variety of specialized daisy wheel character sets, all with appropriate proportional spacing and hammer force. Manufacturers usually offer a slower 30 or 40 character per second model and a faster 55 cps version.

There are some very inexpensive letter quality printers such as the TP-1 from Smith-Corona, but be forewarned that these machines are called letter quality only because they print fully formed characters. The printouts look like they were typed on an inexpensive portable typewriter, the machines lack all of the typesetting features (proportional spacing, superscripting and subscripting, etc.), are often very noisy and at 12 characters per second drag on about their work interminably. TP-1's have been known to collapse from exhaustion after two or three months of light work; so they really don't save you any money. The CR-II from Comrex (see Figure 19.5) is a pleasant surprise in this regard. It is an inexpensive, 12 cps daisy wheel printer that is reliable and produces good quality text.

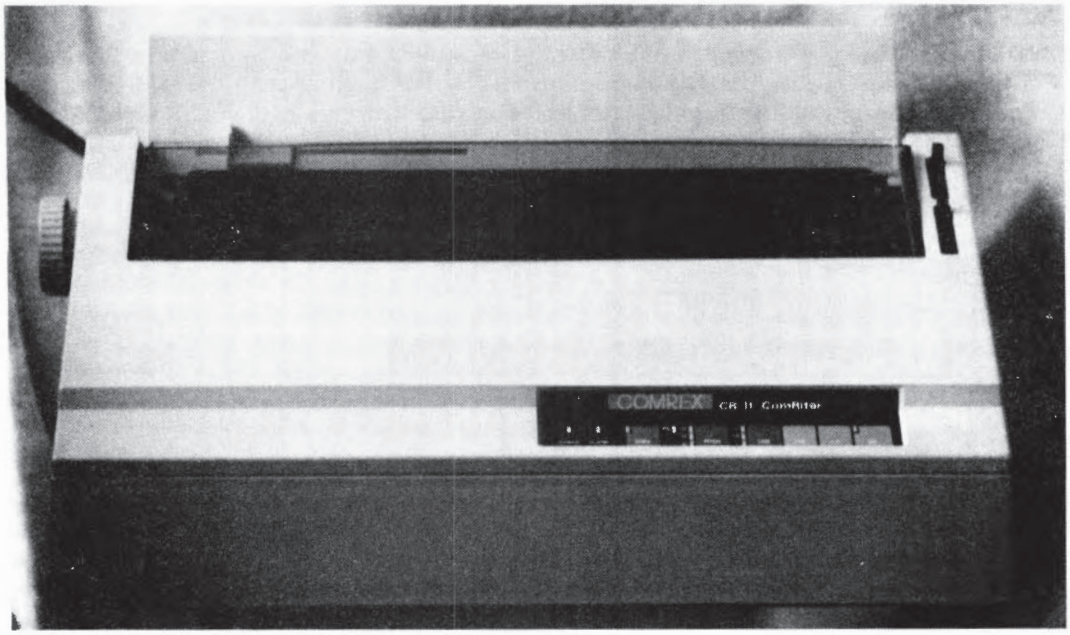


Fig. 19.5 Comrex CR-II low cost, low speed letter quality printer.

Color Printers

Color printers are fairly similar in price to standard dot matrix printers. The color printers from IDS and Transtar both use multicolored ribbons and a lot of fancy maneuvering with the print head. The Scribe printer from Apple uses a new heat transfer technology. The pins in the head heat up rather than snap forward. The ribbon has a wax-like base which responds to the heat by melting some ink onto the paper (any kind of paper). Canon has produced a much more reliable and much faster alternative in its A-1210 Ink Jet color printer. This machine is quiet, produces very crisp, impressive color graphics, and comes in with an astonishingly low price of $635.

One serious problem in computer color graphics has been the difficulty of reencoding a high resolution color video display so that it can be sent to a color printer for duplication. This is

an especially serious problem for the new very high resolution color video systems such as the Number Nine Graphics Board (see Chapter 7). As a result, such images are often just photographed off the video screen.

New color ink jet printers from ACT and Envision Technology provide an interesting solution. You simply plug your video output into the printer. These printers contain decoding circuitry and large amounts of RAM display memory which are used to recreate the bit pattern of the video image. The printer then applies its own software to reencoding the image from its display RAM into a pageful of printing commands.

Plotters for Professional Graphics Printouts

Most formal illustrations in business, science and commercial art must be drawn out with continuous lines and solid color fills so the "dot graphics" abilities of dot matrix printers and letter quality printers are completely inadequate. Plotters are machines which do exactly what graphic artists have always done with pens and cover overlays. The enormous advantage of a plotter over standard drawing is the potential to rapidly edit and change an otherwise beautiful and complete drawing. Graphic design with a plotter requires all the aesthetic skill of traditional drawing, but the techniques for executing the drawings involve skill with BASIC programming or the use of a new generation of "picture processing" software.

Plotter Graphics versus Dot Matrix Graphics

Computer graphic design with plotters also differs from dot graphics in that is often done independently of the video graphics of the computer. Apple high res video graphics has a resolution of 280 dots across about eight inches of your monitor screen so it comes out to about

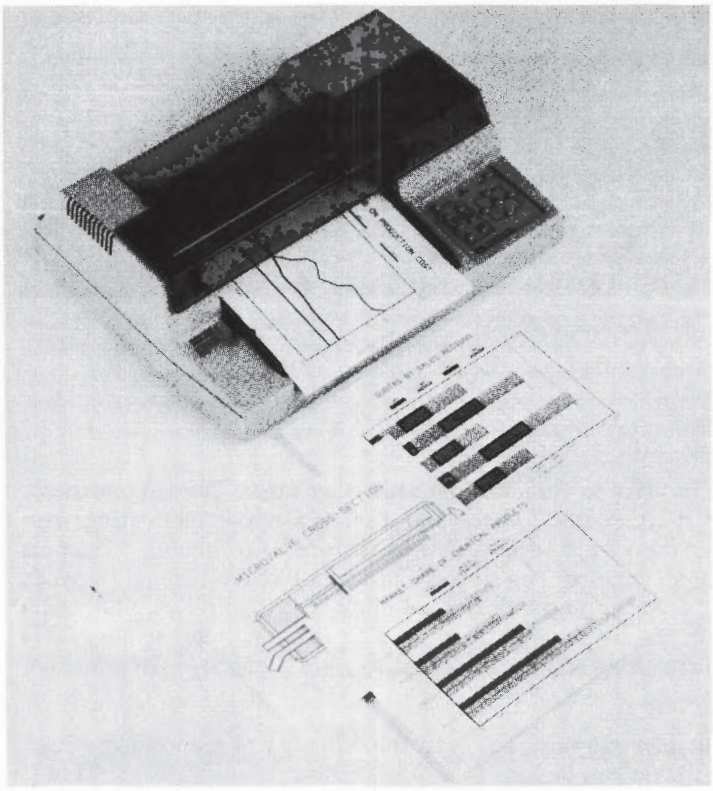


Fig. 19.6 Plotters offer a sharp increase in graphic resolution relative to dot matrix printers. This HP 7470A is operated by the standard HP Graphics Language.

35 dots per inch. The Hewlett Packard 7470A plotter has control at a resolution of 1000 units per inch, all in smoothly drawn connection.

Drawing out graphics on a plotter can be simpler than designing video images with Applesoft BASIC commands, and it is infinitely simpler than doing graphic design with a dot matrix printer. Although the high resolution Epson FX-80 dot matrix printer can generate graphics with a density of 240 dots per inch, a program must be able to calculate the position of every dot on the page and send commands to fire individual pins at specific locations at millions of locations on a page.

This is not impossible, but it is difficult and there is little in the way of practical software available to do this. It is for this reason that most graphics on dot matrix printers is done through "screen dumps" where the computer calculates dot positions on the video screen using complex machine language programs built into the Applesoft Language Interpreter, and then the dots are copied out to the printer as the print head does its sweeps across the page.

Plotters are much simpler to program because they accept what are called "vector graphics" commands. A simple command to draw a line from (0, 0) at the top left to point (7,500, 7,500) at the bottom right causes the pen to zip across the page in a smooth straight line from the first point to the second. To execute the same command on a dot matrix printer would require the calculation of tens of thousands of individual points along the way, and none of the dot matrix printers are able to do the calculations for you.

You can cause a dot matrix printer to generate smooth, well formed graphics if you use a high powered computer such as the Lisa, or with an Apple II if you install a special high resolution graphics board which uses the NEC 7220, a powerful vector math graphics microprocessor (see the description of the Number Nine Graphics System in Chapter 7). A good plotter will do all this work for you with simple convenient commands (see Figure 19.6).

The Plotter Software Problem

Plotters are oriented towards vector drawing and high resolution, but most of the graphics software on the market is oriented towards dot graphics on low to medium resolution video screens; you must therefore either purchase special software for your plotter or be prepared to write your own BASIC programs to operate the plotters.

Each of the plotters on the market seems to have its own fairly limited set of commands. Because the various command sets are incompatible, the market for plotter software is divided into several small segments and this means there isn't much software and that there may never be very much. The limited set of commands on most plotters makes the task rather difficult because a fairly large number of steps are required to construct complex objects.

The Hewlett-Packard Graphics Language and the 7470A

As microcomputer owners started buying plotters and started running into the software barrier, some began to notice that there is something special about Hewlett-Packard plotters in the software realm. Because HP has been heavily involved in advanced graphics and plotter systems for many years, there is a fair body of software written in BASIC using a set of commands referred to as Hewlett-Packard Graphics Language or HP-GL. Further, if a businessman had purchased an HP plotter for an Apple, it was often possible to walk down the hall to the company's computer graphics lab and find a programmer who was "fluent" in HP-GL.

Hewlett-Packard has recently noticed just how interested microcomputer owners are in their plotters and has responded by launching an ad blitz in the microcomputer magazines and by slashing the price of their extremely powerful 7470A from $1600 down to a reasonably affordable $1095. Other plotter manufacturers have responded to the software problem by including only six or eight vector commands and three or four commands for drawing alphanumeric characters (Sweet-P, Amdek DXY-100, Strobe-200) apparently in hopes of making their machines simple to program. Meanwhile, HP has maintained a powerful and expressive set of 20 vector graphics commands (including circles, arcs and special line patterns) which are drawn out at 15 inches per second, and 15 commands for controlling labeling with numbers and letters (including five different international and scientific character sets) which get drawn at six characters per second; as good as a fast typist.

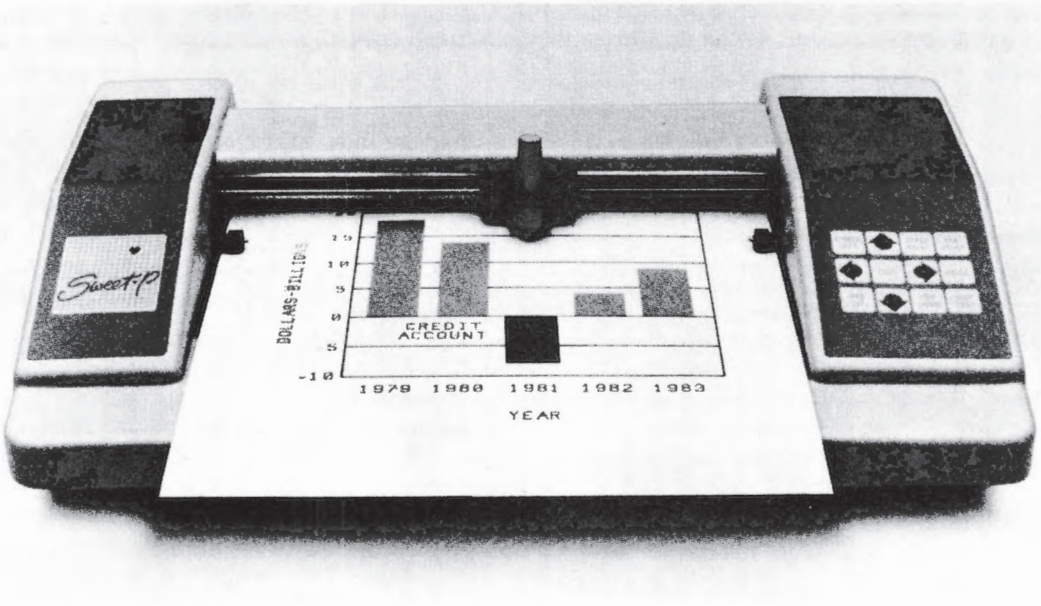


Fig. 19.7 Sweet P plotter from Enter.

The HP 7470A also parts company with the other plotters in that it provides 10 commands which cause the plotter to report back to the computer. These can be used to operate the plotter as a crude digitizer, but are also very important for editing drawings. Owners of an HP 7470A have the broadest selection of available software, and are likely to be able to find graphics programmers familiar with their device; but if they decide to program on their own, HP includes a really excellent 200 page programming guide and tutorial.

The Sweet-P from Enter Computer, Inc. ($795; see Figure 19.7) has one pen, a simple command structure, and a resolution of 250 units per inch, but it permits its Y-axis to run out continuously along a 10 foot roll of paper for strip chart recorder simulation. The Amdek DXY-100 ($749) can handle 10 inch by 14 inch paper and can be enhanced with an extra ROM chip to provide circle and arc commands. The Strobe-200 ($730) has fairly good resolution (500 units per inch), and a respectable software base, but it is fairly slow (three inches per second) and has an extremely simplified command structure. The Apple Color Plotter Model 410 (see Figure 19.8) draws at just four inches per second, but it is software compatible with the HP 7470A.

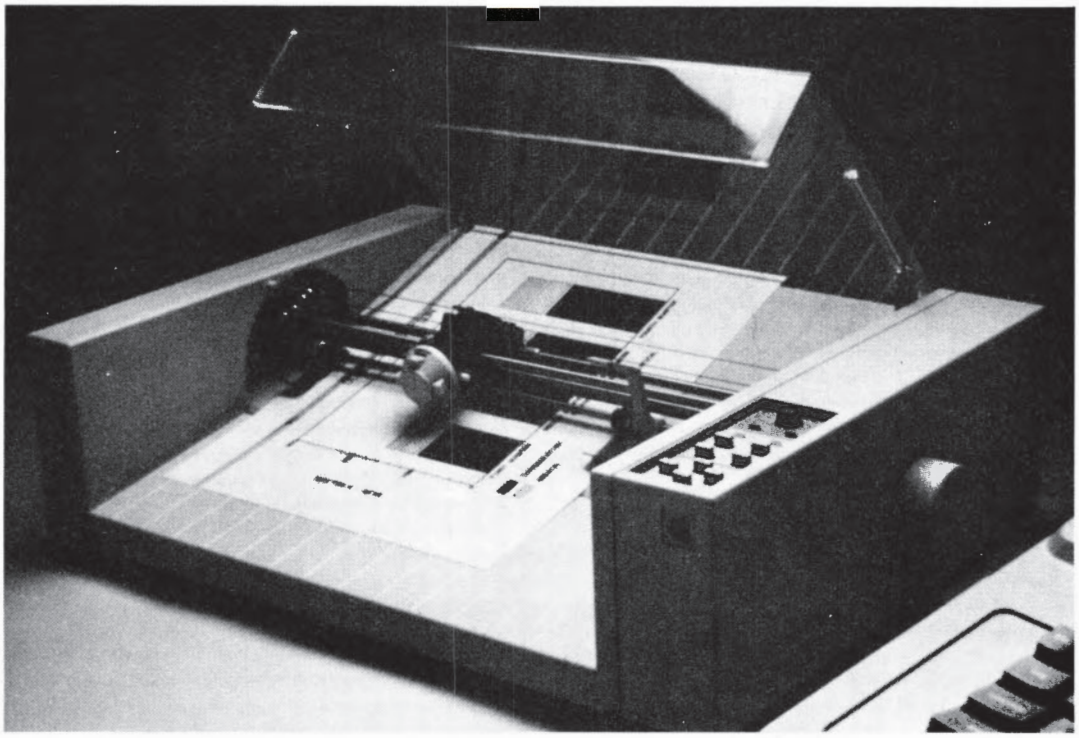


Fig. 19.8 Apple Model 410 plotter.

Changing Pens to Get More Colors

All of the plotters on the market for microcomputers have just one "drawing hand," so they must change pens to draw different colors. If you are using a Sweet-P, an Amdek DXY-100, or a Strobe, you must first do part of the drawing in one color, then cause the machine to stop while you change to a different color pen before permitting it to continue. The HP 7470A has a built-in holder for a second pen, so you can write pen changing commands into your program. To get more than two colors, however, you must be present near the machine to switch the pens at the right moment.

Particularly with slower plotters, the need to stand about waiting so you're present to do the pen changes when necessary can be very tiresome. Several manufacturers therefore offer plotters with several pens and commands which make the plotting hand stop, put away its current pen, and then pick up a pen with a different color before proceeding. Bausch and Lomb (Houston Instruments) offers an eight pen system (the DMP-29, $2295), Strobe has a six pen machine, and Amdek offers a fairly versatile six pen plotter for $1299. Hewlett Packard has just released a six pen plotter ($1895; see Figure 19.9), but software must be enhanced to take advantage of this additional feature.

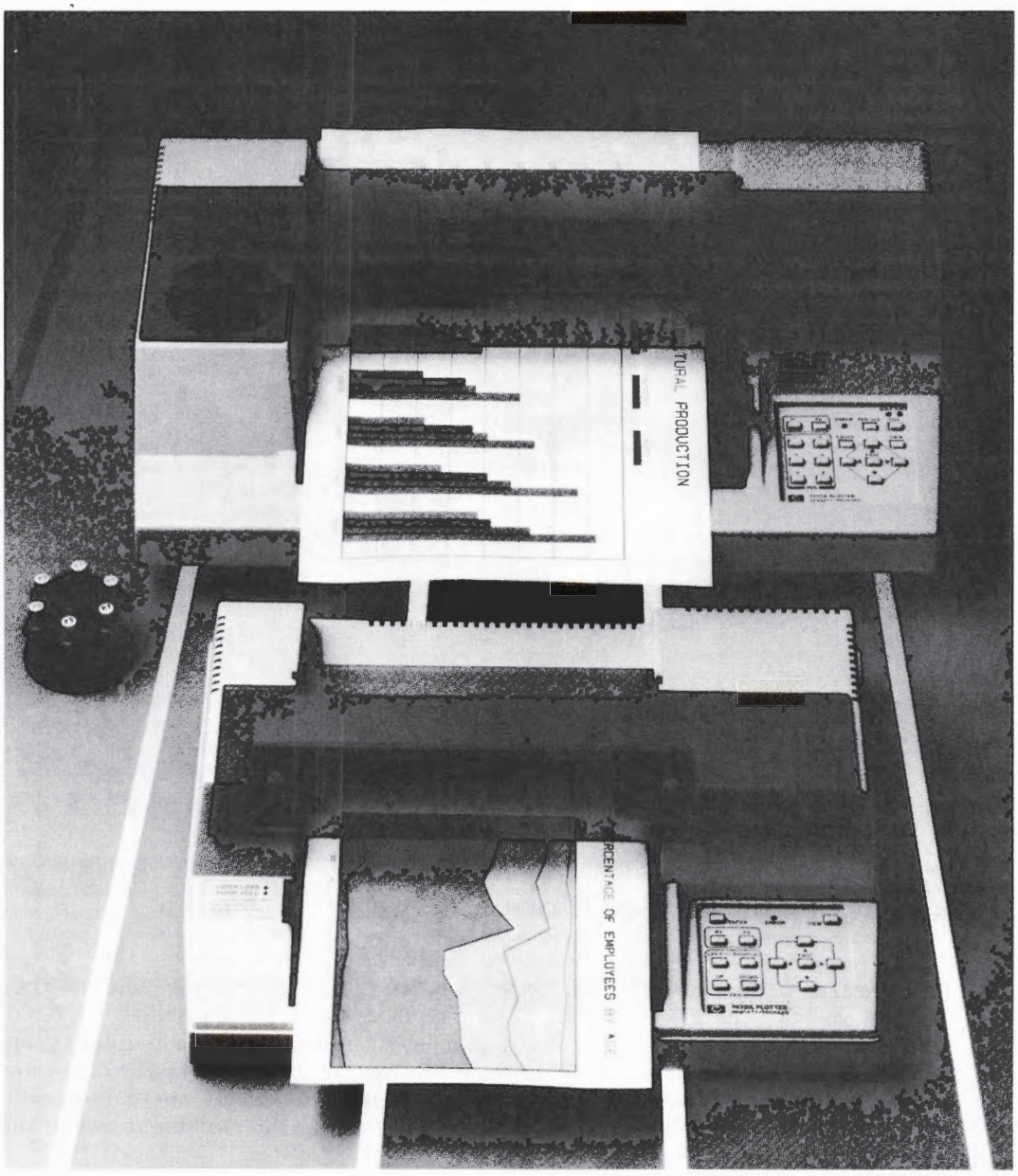
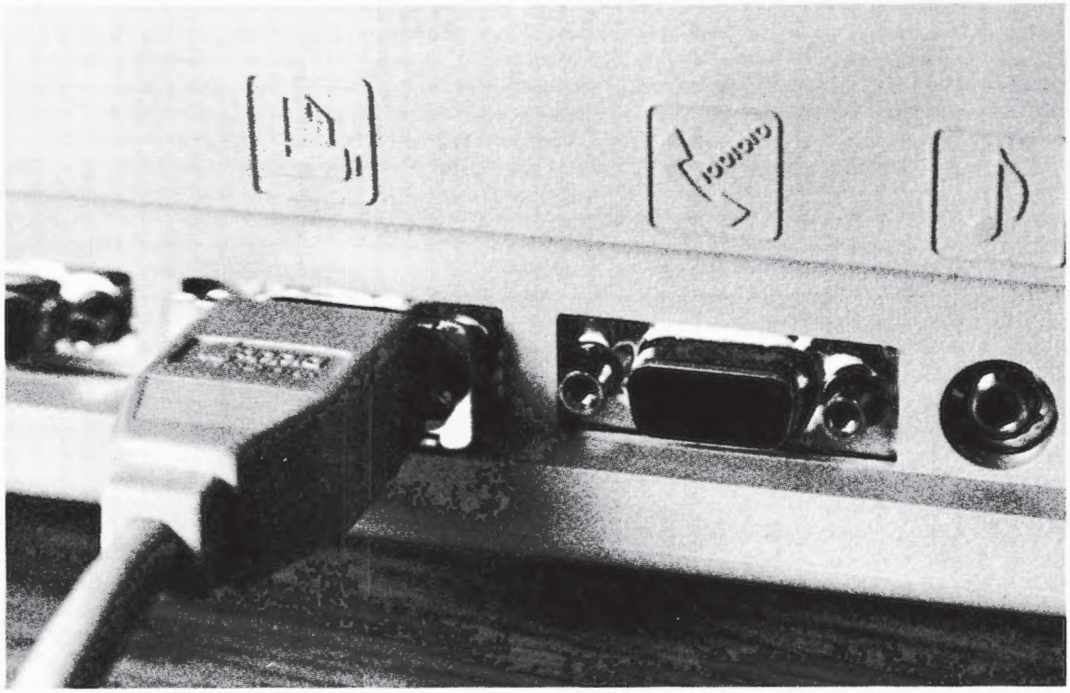


Fig. 19.9 The HP7470 (front) can switch between two pens, and the HP7475 (back) has six pens.



Chapter 20

Local Area Networks and IEEE-488

More Digital Connection Systems

Speed and Distance for Networks

Most of the discussion of communications so far has been taken up with a review of the RS-232C and the Centronics Parallel connection systems. These two systems are just great for connecting one modem to one computer or one printer to one computer, but there are a whole range of other and very important communication functions at which these two fail miserably.

The three great failings of RS-232C and Centronics Parallel are in the realms of distance of transmission, speed of transmission, and participation of more than two devices. Distance is not a big issue when you're connecting a printer to an Apple, and that is very fortunate because Centronics systems cannot manage secure transmission at distances over three or four feet. RS-232C is a little better in that you can probably run a cable along the wall to another machine on the other side of a big room, using up to 50 feet of wire, but that's it.

Similarly, it's no big deal if you can't use really high speeds to send data to a printer or a modem because, after all, printers just can't type all that fast, and, as reviewed above, telephone lines only accept very slow transmissions. Let's be clear about the meaning of slow. The 6502 shuffles data around at a rate of over five million bits per second, and the Apple's video display generator pumps information at just over 75 million bits per second. A very fancy Diablo printer accepts data at 300 bits per second. That is what is meant by slow.

If you want to connect two Apples which are 60 feet apart (as the cable lies), and you are willing to put up with a 99.99 percent reduction in speed to 300 baud, you're still going to have to spend $300 or $400 on modem equipment. You may be thinking that there must be another way, and indeed there is. Actually, there are several other ways.

Age Old Current Loop Returns for the Distance

The mention of a "20 milliamp current loop" conjures up images from decades long gone by of old mechanical teletypes tapping out characters on long rolls of yellow paper and blowing air through holes in paper tape to read stored data. But consider this: the 20 mA current loop

system uses the same asynchronous serial protocol that is so popular for RS-232C systems, but a 20 mA current loop can transmit at 9600 baud at distances up to 1500 feet. As a consequence, 20 mA current loop has recently been experiencing something of a small scale renaissance.

Among the factors which limit the length of RS-232C cables, the two chief culprits are "capacitance" (see Chapter 13), which slows the transition from -12 volts to +12 volts, and "ground loops," which cause confusion about absolute voltage levels; both factors tend to get worse with distance. The 20 mA current loop system is effective over long distances because a strong current can be made much more resistant to capacitive effects as well as to voltage drops due to resistance in the wire. The ground problem is much less serious because 20 mA current loop does not depend on measurements of voltage for detection.

The old Apple Serial Interface Card includes a 20 mA current loop interface, so for $80 you can have two Apples communicating at respectably high speed at opposite ends of one awfully large building. If you spent $1000 to buy a couple of 1200 bps modems, you'd be spending 10 times as much money to get one tenth the communication speed. As if the economics weren't impressive enough, it's nice to know that the protocol is logically identical to what is used for data encoding in RS-232C systems so all of the standard, off the shelf communications software will work just fine.

Another reason to keep 20 mA current loop in mind is that there are still tens of thousands of lab instruments out there feeding data into teletypes, and equally many undergraduates copying the information off of the yellow paper in order to type it into the laboratory Apple for analysis. Instrument manufacturers who would like you to spend many thousands of dollars for the new "smart" version of their machine don't seem to be too helpful on this point. However, there is no need to continue in this fashion because it should be no secret that any machine which will talk with a teletype will talk just as happily with an Apple.

A simple Apple card you can build to do the RS-232C/20 mA conversion is shown in Chapter 13, Figure 13.2 in case you already own a fancy serial card, don't want to replace it with an Apple SIC, and want to add current loop capability cheaply. The actual transmission distance for a given current loop system depends on the nature of the current source. A 1.5 volt battery and a 75 ohm resistor yields a 1.5/75 = .02 amp current, but the 12 volt Apple supply with a 600 ohm resistor generates a "stiffer" version of the same 12/600 = .02 amp current. A long wire which adds 100 ohms of resistance is rather rough on the first current source, but has comparativley little effect on the current from the second source. Consult your local engineer for details on long line current loop connections.

Fifteen Devices On-line with IEEE-488

The current loop system just described offers a 30 fold increase in transmission distance over RS-232C, (and 500 fold increase over Centronics Parallel), but it offers no real improvement in speed over RS-232C, nor does it permit more than two devices to be connected together. A completely different system called IEEE-488 operates over just slightly greater distances (65 feet) than the RS-232C system, but it permits communication at up to a million bits per second, and provides a protocol which lets 15 different devices talk among each other on the same interface cable.

The IEEE-488 is a fairly elaborate parallel communications interface which was born of the concerns of manufacturers of "smart" laboratory instruments and industrial testing equip-

ment. The typical setup for an IEEE-488 system would be a room with two or three microcomputers and a dozen fairly intelligent testing, monitoring and analysis instruments scattered about performing automatic measuring chores.

Controllers, Talkers and Listeners

In this system, any of the devices, including the microcomputers, can send information to any other device. Only one can speak at a time, but any number can listen. Although this system provides for very rapid transmission of data, it must be configured before each round of communication, and the configuration process is a bit clumsy.

At any given moment, just one of the devices is considered to be the "bus controller." There are 16 wires used in the bus, eight for passing data and addresses and eight used for a variety of control and signaling functions. Whenever one of the devices has some information to send, it places a 5 volt signal on one of the control lines called Service Request (SRQ). The controller can respond to this request by configuring the system for a little communication.

Configuring for Communication

To configure the system for a transmission, the controller puts a signal on a special control wire called Attention (ATN) and all the devices respond by sort of tuning in to find out what role they are supposed to play. Each of the devices has its own permanent address, and the controller is able to load these addresses onto the eight wire data bus part of the IEEE-488 cable. For communication, the controller addresses the devices one by one to find out which one pulled the SRQ line. When it finds out, it designates that device the "talker." The controller can either listen alone to what the talker has to say, or it can address several other devices and tell them that they will all be listeners.

Talking and Listening

Once the bus is configured, ATN is turned off and communication can begin. The designated talker now takes control of the data bus. All of the designated listeners are supposed to signal when they are ready to receive data. They do so by trying to put a 5 volt signal on a control wire called Not Ready For Data (NRFD), but that wire is set up electrically so that it will not respond until all the listeners have attempted to put the signal on. The talker waits for NRFD to change, and when it does, it puts a byte of information on the eight wire data bus and puts a signal on a control wire called Data Valid (DAV). All the listeners turn off their NRFD signal, and as each one captures the data, it tries to respond by putting a signal on another control wire called Not Data Acknowledge (NDAC).

The sequence in the preceeding paragraph is repeated until the talker is finished with its message which it indicates by putting a signal on another control wire called End Or Identify (EOI). When the controller sees EOI, it knows it can go about looking for new SRQ request signals.

For those who are interested in a more complete explanation, there is a very nice discussion of this system in Part 3 of Steve Leibson's I/O primer (Byte: April, 1982, page 186) and a somewhat less concise discussion in *PET/CBM and the IEEE 488 Bus (GPIB)* by Eugene Fisher and C.W. Jensen (Osborne/McGraw Hill).

Apples and IEEE-488

Although all the fiddling around described in the previous few paragraphs sounds quite complex, it all takes place at a very good clip. The result is a fast and versatile system which has enjoyed considerable popularity among instrument manufacturers. It can be used to provide a comparatively inexpensive way of linking several nearby Apples (no two can be more than 12 feet apart, although the entire cable may be 65 feet in length). Communication takes place at high speed and it is fairly easy to add devices to or remove devices from the system. Nonetheless the primary use of IEEE-488 interconnections will be for connections with equipment which can only speak in IEEE-488 format.

There are two very different kinds of IEEE-488 boards available for the Apple. A very elegant and easy to operate board is made by SSM and there is also an easily programmed IEEE-488 board available directly from Apple. The alternative is a low cost rough cut system from CCS. The SSM board uses a special integrated circuit from Motorola, the 68488, to manage most of the details of talking and listening, and also provides a sophisticated control program in firmware.

The SSM A488 accepts a set of English commands which can be sent from BASIC, FORTRAN or Pascal and reports results in English words. Since the various devices attached to the IEEE-488 lines were designed without knowledge of the Apple's idiosyncrasies, SSM has provided buffering and data reception features which permit the Apple to accept a variety of otherwise forbidden characters, and to accept strings of 1000 characters although BASIC is usually limited to 256 characters. The bad news is that the board lists for $475.

A less expensive alternative is a sort of stripped down IEEE-488 interface from CCS, the 7490. This card depends on several generalized chips to carry out the interface, so a great deal of the firmware is devoted to housekeeping tasks done automatically by the 68488 on the SSM card. As a result, programming the 7490 is a far more difficult task best undertaken by very experienced programmers. The primary advantage of the CCS card is a much more attractive price tag of $200.

Local Area Networks

The design objective for the ultimate Local Area Network is nothing less than permitting the microprocessor in one microcomputer to send data to RAM chips in hundreds of other computers scattered over several miles at the same speed that it speaks to its own chips a few inches away. In this ultimate system, all makes and brands of computers communicate freely among each other without regard for differences in operating systems, applications software, or company of origin. Communications should be secured for authorized users only, highly resistant to noise and interference, and sufficiently flexible to permit the addition or removal of microcomputers from the system without any interruption whatsoever.

This is one tall order.

There are a few obstacles impinging from the real world. For instance, during one clock cycle of the Apple, a beam of light can only travel about 1000 feet. Even more obstinate than the speed of light is the obstacle provided by computer manufacturers who depend on incompatibility to encourage you to continue buying their company's type of computer.

With these obstacles in mind, it is a remarkable fact that most of the elements of the ultimate Local Area Network exist and are in place in some American corporations, and the full system should be operational and available very soon.

Network Architecture: OSI and SNA

The design strategy for this daunting engineering task has been to break the problem down into a series of simpler elements. Each element, or "layer," has been attacked as a distinct problem, and then "interfaces" have been worked out to connect the layers together into a complete system.

This breakdown into layers has been done for the purpose of making it easier to pin down design problems, but the terms used to describe the layers are still very helpful for someone approaching networks for the first time. The term used to describe a layering scheme is "architecture." There are two major views about the architecture of a network system, one is the widely accepted product of the International Standards Organization (ISO) and it is called Open System Interconnection (OSI). The other major view of network architecture comes from IBM and it is called Systems Network Architecture (SNA).

The OSI architecture has had enormous impact in the design of nearly all local area network systems. The SNA system, however, has had a fairly negative impact so far, because, like the OSI system, it is not yet completed in workable form, but, unlike OSI, it is a corporate secret at IBM. Many manufacturers and users have postponed work on or purchases of OSI based systems (such as Ethernet) in the knowledge that the impending release of the IBM system must be taken into account. Nonetheless, as the years have gone by OSI architecture networks have crept into thousands of facilities.

The Seven Layers of OSI

As befits a daunting engineering task of heroic proportions, the ISO architecture is universally discussed in the terms of a sort of mythical catechism called the Seven Layers of OSI.

First is the Physical Layer, which determines the substance of the actual connections, next is the Data Link Layer, which states the protocol for coding the data, and third is the Network Layer, which addresses the "topology" and the routing of packets of information. These first three layers are the only ones for which standards exist and for which compatibility can even be talked about.

The four higher layers are the Transport Layer, which permits a given member computer to supervise the dispatch and arrival of completed messages, the Session Layer, which attends to Log-On and data security, the Presentation Layer, which funnels information from, i.e., one company's word processing program into a lingua franca understandable by all, and finally the Applications Layer, which provides for software such as electronic mail programs whose sole purpose is to let the user interact with other people by taking advantage of the services provided by the six layers below.

The Physical Layer: RS-422A, Coax and Optical

The physical layer determines the speed and distance limitations of the network and can play a large role in determining costs. The speed and distance properties of RS-232C are completely inadequate, so network designers have looked to the successor of RS-232C, called RS-449.

Like its popular predecessor, RS-449 includes a large number of control wires and the number of pins is now boosted to 46 in a 37 pin and a nine pin connector. Fortunately, it is possible to do away with 44 of these wires and use only the two data wires for network links. The committee that designed RS-449 foresaw this limited use and wrote a partial standard called RS-422A which describes only the properties of the data wires.

In a RS-422A system, data can be transmitted at a rate of one million bits per second (1 Mbs) and the cable may be 2000 feet in length. The minimal configuration sends the data along two wires at the same time. The two are twisted around each other (thus the name "twisted pair") and the improved speed and noise resistance actually results from this pairing system. In the old RS-232C system, signals were measured as the voltage difference between the TD line and a ground line, but the voltage of the ground line couldn't be depended on for long distances. In RS-422A, the signal is sent as a voltage difference between the two wires, so local ground problems can be ignored. The twisting also helps limit noise broadcasts from the wire which could interfere with other equipment.

This twisted pair system is fairly inexpensive to lay out, and is sufficiently flexible to bend around corners and such, although costs go up if you need to run the wire inside conduit to minimize noise. A more expensive alternative is to use coaxial cable where the signal and the noise shield are built together into the wire.

Coaxial cables can carry signals at 10 Mbs and can be extended over greater distances than RS-422A twisted pairs. Coaxial cable is a little more difficult to lay out and costs more by the foot, but since conduit is not required and its performance is better than RS-422A, "coax" has become extremely popular for networks and is used in Ethernet.

One way of completely eliminating the problem of electrical interference and noise generation is to not use electricity in the connecting cables. Inexpensive lasers and photodectors can be fabricated as silicon chips, and when they are connected by top quality optical fibers the performance is very impressive. Transmission speeds can be in the hundreds of millions of bits per second, and fibers can be many kilometers in length. Few computers can pump information that fast, so optical fibers permit "multiplexing" of several different signals into the same cable at the same time. The chief limits on current optical technology have to do with problems in splicing multiple connections. Reception and retransmission at each node pushes up cost and cuts down on speed, so coax still dominates for high speed systems.

The Data Link: Synchronous Serial and HDLC

It is the responsibility of the Data Link Layer to actually assemble the address, control, and data information into a stream of bits, complete with services for error detection and correction. The data link layer must figure out when an error has occurred and request a repeat transmission of an information packet. In a sense, when this layer does its job correctly it makes the data transmission lines appear error free. The Network Layer hands down information to be sent and receives complete decoded characters back from the Data Link. Noise and interference are detected by the Data Link Layer, and the Network Layer does not have to get involved in such trivia.

Modem communications with RS-232C connectors usually use a coding scheme called "asynchronous serial" (see Chapter 16). In that system, a slow "baud rate" is used and the sender and receiver are expected to stay in synchrony for only 10 or 11 bits, and then to try again when the next character is ready to go.

All of the networks use a very different system in which the sender and receiver are expected to get into synchrony and maintain timing while thousands of information bits pour in. This sort of "synchronous serial" communication requires a completely different system for establishing speed, for establishing the beginning and end of data frames, and for error checking.

Older systems used a synchronous serial communications standard called Bisynch in which ASCII control characters were placed at the beginning and end of a data stream to handle all the management problems. Newer systems have achieved better performance and improved compatibility among different kinds of computers by abandoning the "character orientation" of Bisynch. The newer protocols treat the entire information stream as a bit pattern and do various kinds of fiddling with the ones and zeros to encode synchronization, address and error checking abilities. One version, called Serial Data Link Control (SDLC), was developed by IBM, but the ISO has adopted a modified and more flexible version called High Level Data Link Control (HDLC).

In HDLC, each "message frame" begins and ends with a "flag field" which is a special bit pattern used both as a marker and to assist in synchronization. Immediately following the flag field is an "address field;" in SDLC, the address can only be eight bits in length, so only 256 devices can be in the system. In HDLC, the address field can be as long as necessary. Ethernet takes advantage of this to permit 1024 devices to be on the network and to permit the addressing of several different locations within a given device.

The address field is followed by a "control field" which lets the receiving station know something about how to interpret the bits in the information field. The information field can contain any number of bits in any code, limited only by the memory capacity of the receiving device. The information field is followed by a "frame check" field which contains a mathematical abbreviation for the contents of the information field. The receiving station recalculates this check value from the incoming information and compares it to the value in the frame check field. If the two numbers are different, than a transmission error must have occurred and the receiver sends a message back to the sender requesting a retransmission. For a more complete discussion of the various protocols see *A Manager's Guide to Local Networks* by Frank Derfler Jr. and William Stallings (Spectrum/Prentice Hall) or *An Introduction to Microcomputers, Volume 1* by Adam Osborne (Osborne/McGraw Hill).

The Network Layer: Stars, Rings, Buses and CSMA/CD

The three popular "topologies" for networks are shown in Figure 20.1. The star arrangement implies a central controlling unit, and although this is the model for telephone PBX systems, it has been unpopular for networks because a failure at the center wipes out the entire system. The ring system's greatest problem is that the whole network must be shut down whenever any machine has trouble or whenever one machine is added or removed. It provides some advantages for determining who will speak and when, and it is expected to be the basis of IBM's SNA network system.

In the ring configuration, one member device is able to seize control of the ring so it can send packets of information without interruption from others. A "token" pattern such as "1111 1111" is passed around the ring from one device to the next continuously until one of them has something to say. It siezes control when it receives the token by sending out a modified pattern such as "0111 1111." All the other devices are restrained from sending messages and can only pass along the changed token and interpret addresses until the sender changes the token back to the all clear form.

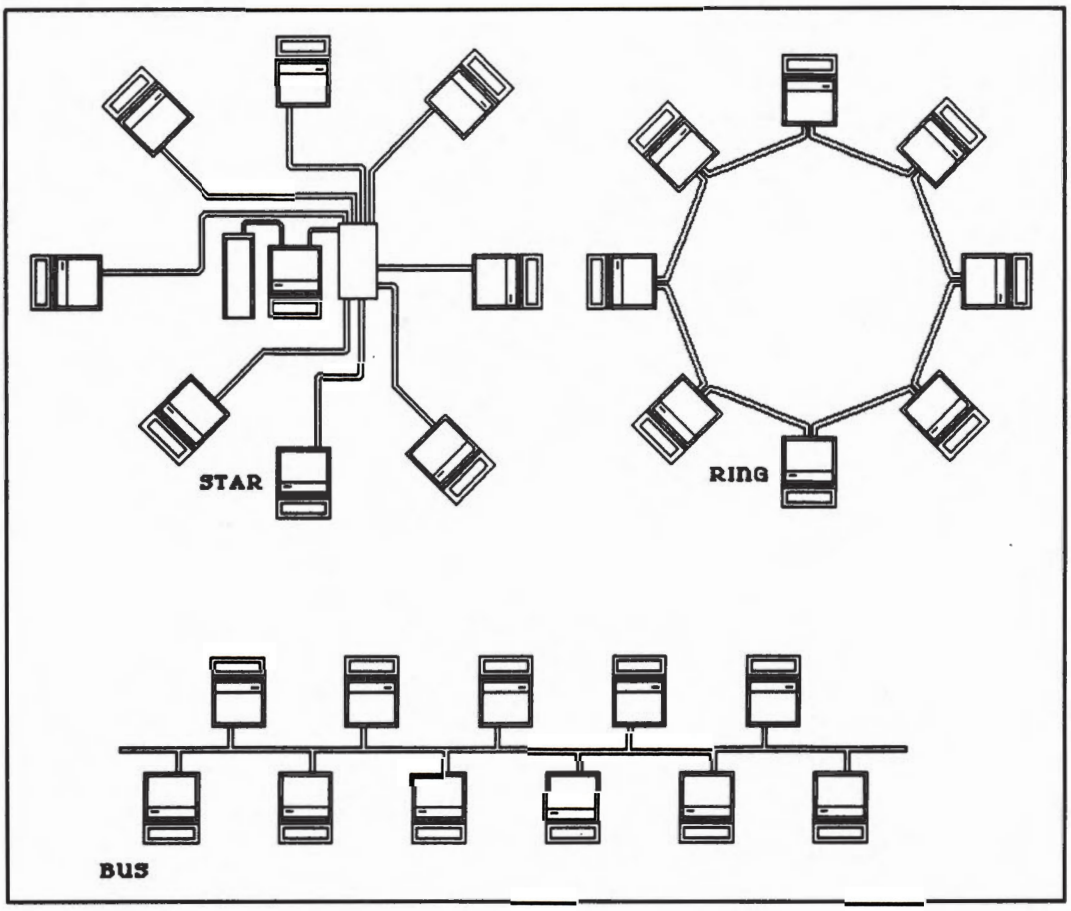


Fig. 20.1 Network Topologies.

The bus system is preferred in the ISO system and is used in Ethernet. The outstanding advantage of the Ethernet bus system is that all communications continue at full speed when computers are added and removed, and a failure of one device has no effect on the system. The address scheme is "dynamic" so that it can adjust to the addition or removal of devices.

The trick in a bus system is to ensure that only one device will be speaking at a time. Nearly all bus networks use a system called Carrier Sense Multiple Access (CSMA) to maintain order on the bus. There is no controller, no master, and no priority system. All that is required is that each member take a quick listen before it tries to send. If it senses a "carrier" (meaning that someone else is talking), it waits and tries again. Since a full CRT screenful of text takes about 1/500th of a second to be transmitted at Ethernet speeds of 10 Mbs, the line ends up not being used most of the time, and waits are rarely very long, even with hundreds of computers on line.

One situation which is not handled very well by CSMA is when two devices attempt to begin a message essentially simultaneously. Xerox has a "collision detection" system in which the interference pattern from simultaneous access can be detected, both senders halt and then attempt to restart after an essentially randomized interval. Bus systems with a full collision detect mechanism (CSMA/CD) are usually more expensive than simple CSMA, and collisions are fairly rare so many smaller systems do not include collision detect facilities.

The Higher Layers

There are few standards for the higher layers, but the three dominant players are Transmission Control Protocol (TCP) from the Department of Defense, Xerox Network System (XNS), and BX.25 from Bell. These protocols attempt to provide a uniform software environment for the transport, session and presentation layers. At present, the use of Ethernet for the lowest three levels has been endorsed by Xerox, Digital Equipment Corporation, Intel, and a number of other manufacturers, and Xerox's newer XNS for layers four through six is gaining increasing acceptance.

The price of an Ethernet/XNS system has begun to decrease with the development of specialized integrated circuits to handle the interface. Another important factor is 3COM's introduction of Ethernet/XNS interfaces for IBM PCs, Lisa, and Apple II. There have been various rumors about Apple Computer's "Applenet." Many advance reports suggested an RS-422A Applebus twisted pair system, but Apple seems now to be waiting for IBM's SNA.

Another important player is ARCnet from Datapoint Corporation. This system uses a unique network topology which is a sort of hybrid between the bus and the star system. The use of stars to connect buses permits an ARCnet system to sprawl over four miles. It does not use the same kind of coax cable as Ethernet, so its transmission speed is a slower 2.5 Mbs, but its cable is directly compatible with the cable used in IBM 3270 systems.

Rather than using a CSMA/CD system like other buses, ARCnet uses a token passing scheme like the one described above for ring networks so it requires a comparatively simple interface card. Nonetheless, ARCnet uses XNS for its higher layers so it offers software compatibility with Ethernet systems. The Nestar PLAN 4000 is essentially an ARCnet system which provides interfaces for Apples and IBM PCs. A Nestar interface costs about $400 while an Ethernet interface still costs nearly $1000.

The third major network system for Apples is Omninet from Corvus. Omninet uses a 1 Mbs, RS-422A system and a fairly intelligent interface card. The Omninet system has its origins as a successful effort to permit several Apples to share a single hard disk. The interface card has its own 6801 microprocessor which sends and receives message packets and then uses "DMA" (see chapter 27) to move messages in or out of Apple memory. These interfaces cost about $500.

Apple Computer manufactures a 3270 Cluster Controller Emulator which can be used to operate an Apple II as if it were an IBM 3270 terminal. The Apple communicates with the Cluster Controller via a standard serial link, while the controller handles the details of the signal and electrical interface.

There are versions of the controller available for a cluster of four or of seven Apples. "Apple line" can be used to connect to an existing cluster controller from another manufacturer, and there is also a "Protocol Card" to handle details of terminal emulation.

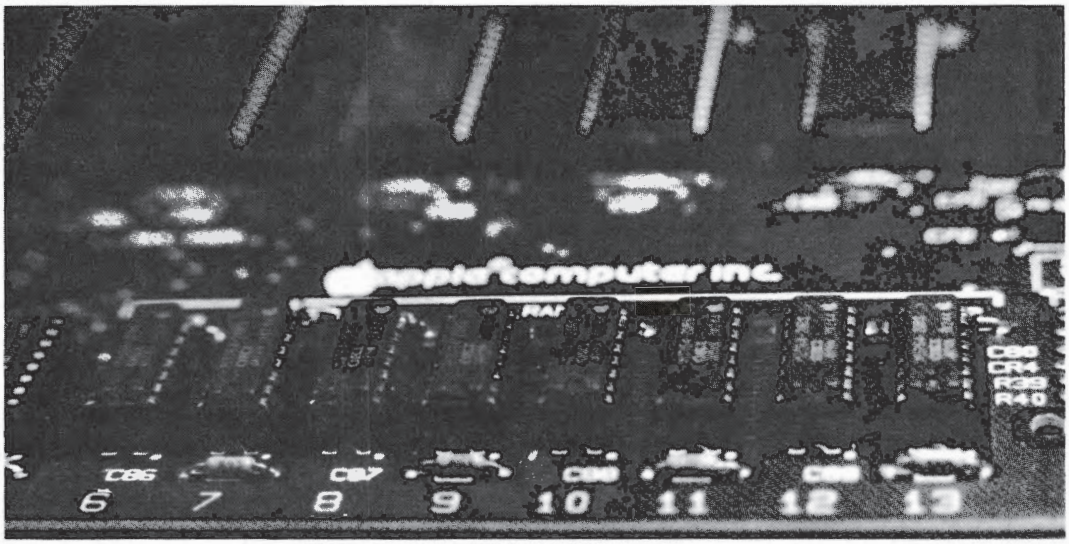
III

PROCESSING APPLE

PART 1

The Addressable Apple

- **CHAPTER 21 Main RAM and High ROM**
- **CHAPTER 22 I/O ROM and Ports in the $C000 Space**
- **CHAPTER 23 Disk Drive Functions**
- **CHAPTER 24 Getting More Disk Capacity**
- **CHAPTER 25 Bank Switching the High 16K**
- **CHAPTER 26 On-Board Ports, Soft Switches and Auxiliary RAM**


apple computer inc.
6
7
8
9
10
11
12
13

Chapter 21

Main RAM and High ROM

The way the Apple uses its memory chips and its disk drives is best understood in terms of three potentially unfamiliar entities: its address space, file buffers and bank switches.

You can appreciate these three things as follows. You sit back in a comfortable chair and put your feet up with no intention of moving for the rest of the afternoon. You're planning to scan a huge mass of books and articles, all with as little muscular effort as possible. God forbid you should have to get up and walk across the room to get something.

To your left is a bookshelf. It is within easy reach of your left arm and you can comfortably put your hand on each and every book and article on the shelf without even having to put down the cup of coffee in your right hand. That one bookshelf plays the part of the Apple's "address space."

This is a class operation, so, in addition to the books on the shelf, you've got a little switch panel towards one end of the shelf. In easy reach, and at the touch of any one of several buttons on the little panel, you can control video, sound, cassette tapes, etc.; a sort of complete home entertainment center. That switch plate represents the Apple's "$C000 space" (pronounced "see thousand"), about which you will learn more shortly.

A single convenient bookshelf is just great, but remember that we were talking about a really huge amount of reading. Still, we will never have to leave our chair. This is all made possible by a couple of very high tech tricks. You see, one book on the shelf is very special because it is sitting in what we will call the "file buffer." The instant you remove a book from that one space, a new one pops in behind it. This file buffer is, in fact, a sort of access chute leading to a potentially enormous library equivalent to about 1000 bookshelves equal in size to your own. The great improvement over going to the library and climbing stairs leading to long aisles of book stacks, etc., is that all the books are available at that one single file buffer space along your chairside bookshelf and they may be just as easily and accurately returned to their proper place in the huge library simply by shoving them back out through the file buffer.

When you begin your work you sit down to a largely empty shelf, containing just the switch plate, one book sitting in the file buffer, and a few permanent old standbys (i.e., the operating manual for your high tech book shelf) off at one end of the shelf (here representing the Monitor and Applesoft ROMs).

Tapping out your selections on the switch plate, the books and articles start showing up at the file buffer. As each arrives, you remove it from the file buffer and put it where you want it on the shelf for subsequent use. Pretty soon, you've brought in say 30 or 40 books and your

shelf is quite stuffed. The problem is you know you just absolutely must have about 10 or 12 more in easy reach. No problem, you reach for the switch panel and hit a "bank switch."

Poof, a bunch of books at one end of the shelf disappears, leaving some nice new empty shelf space. You bring in those last few additional books via the file buffer, put them in place, and tap the bank switch again. Poof, the recent arrivals disappear and are replaced by the ones that had been there first. Just for kicks, you tap the bank switch a few more times, and the two sets of books, both of which you put in that same part of the shelf, exchange one set for the other set, all in order, instantly and en masse. Owners of an Apple //e or //c have an even more powerful set of bank switches and can make the entire set of books along the whole length of the address space exchange with another complete "auxiliary" set in an instant (an "instant" here being defined as a little bit less than one millionth of a second).

A Map of the Apple's Address Space

The address space of the Apple exists as a sort of creature of the 6502. It is defined by the 65,535 (64K) different addresses which can be loaded into the 6502's "address buffer" (see Chapter 27). Once the 6502 is installed, the address space of the Apple has 64K locations whether or not there are any RAM chips on the motherboard. No matter how much more RAM or ROM is added, the address space stays the same; 65,535 locations. As described above, additional RAM beyond 64K must be "bank switched" into the address in order to be used.

This "space" can be represented as a tall thin box which is eight bits "wide" and 65,535 bits "tall." Each byte in the space is a little one by eight rectangle, and the thousands of bits are stacked one on top of another. Because it is a large and complicated two dimensional space, much like the expanse of the Great Plains, the best way to find your way around in it is with a "map."

The "map" of the Apple's address space (which has been drawn in Figure 21.1) has been squashed and distorted in order to squeeze all 65,535 bytes into a reasonable amount of space on the page. Each byte within it is no more than a very thin line. The byte on the bottom line has been labeled 0K and the top byte has been labeled 64K (actually byte number 65,535). Between the top and the bottom of the space, marks have been placed every 4096th byte (labeled 4K, 8K, 12K, etc.), thus dividing the address space into 16 blocks.

Numbering the Bytes

The numbers on the right side of the drawing show the number of each marked byte as it appears in the hexadecimal numbering system. Hexadecimal notation is very important for describing the way computers work. It is called hexadecimal because it uses 16 different digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. This numbering system provides a clear and simple way of marking off the organization of the Apple's memory space because the 6502 can be said to think in hexadecimal, rather than in decimal, as we do.

The trick to using hexadecimal numbers is just to use them without fanfare and not to worry about them. Don't try to convert to decimal unless you have to. It is much easier to remember that the Apple's input/output spaces begin at C000 than that they begin at 49,152. Whenever a hexadecimal number is used, it will be preceeded by a dollar sign ($), i.e., $C000. Some folks would call this number "C thousand" but most seem to prefer "hex see zero zero zero." If you want to be able to convert easily back and forth from hexadecimal to decimal you should

consider buying a Hewlett-Packard 16C computer science calculator. The Apple can do the "hex/dec" conversion for you (see Listing 21.1), but it's often a bother to stop what you're doing to set up the conversion program.

If you refuse to put up with hexadecimal numbers, the rest of us will go along grudgingly. Decimal numbers will be provided in parentheses unless they're going to hopelessly clutter a drawing.

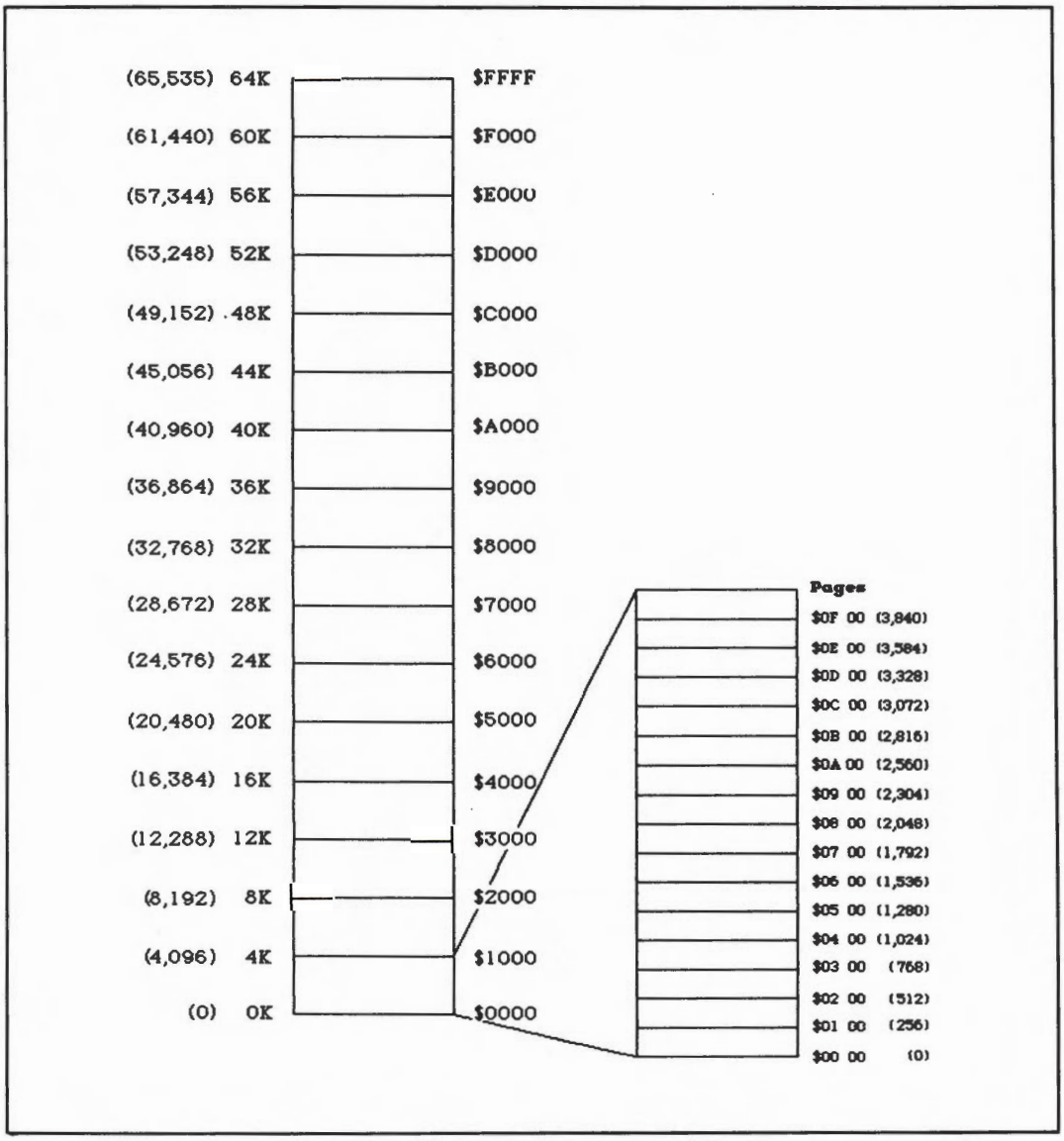


Fig. 21.1 Blank map of the address space. This is a standard way of representing locations in memory. The figure shows 65,535 bytes lying one on top of another. The numbering on the left is in standard decimal, but the numbers on the right are in hexadecimal (base 16) to conform with the internal organization of the computer. The blow up on the right shows 16 pages in the first 4K, each containing 256 bytes.

Hex/Dec and Dec/Hex Conversion

```
10  HOME : PRINT CHR$(12)
20  VTAB 2 : HTAB 5 : PRINT "****** HEX/DEC AND DEC/HEX CONVERTER ******"
30  PRINT : PRINT : HTAB 3 : PRINT "Please type the number of one of the options,
        then hit 'RETURN'"
40  PRINT : HTAB 5 : PRINT "1 - From Decimal to Hexadecimal"
50  PRINT : HTAB 5 : PRINT "2 - From Hexadecimal to Decimal"
60  PRINT : INPUT "";A$
70  IF A$ = "1" GOTO 100
80  IF A$ = "2" GOTO 340
90  PRINT CHR$(7): GOTO 10
100 HOME : PRINT CHR$(12)
110 VTAB 2 : HTAB 5 : PRINT "DECIMAL TO HEXADECIMAL CONVERSION"
120 PRINT : PRINT: HTAB 3 : PRINT "Type any number from -65,535 to 65,535 then
        hit 'RETURN'"
130 X = 5 : PRINT
140 VTAB 9 : HTAB X : GET B$                          :REM * Use GET for commas *
145 IF B$ = CHR$(8) THEN X = X -1: R = R - 1 : GOTO 140
150 VTAB 9 : HTAB X : PRINT B$
160 IF B$ = "-" THEN SB =1: GOTO 200
170 IF B$ = "," THEN B$ = ""
180 IF B$ = CHR$(13) GOTO 210
190 R = R + 1 : B$(R) = B$
200 X = X + 1 : GOTO 140
210 FOR T = 1 TO R: C$ = C$ + B$(T) : NEXT : C = VAL(C$)
220 IF SB = 1 THEN C = 65536 - C : SB = 0
230 IF C > 65535 OR C < 0 THEN PRINT CHR$(7) : CLEAR : GOTO 100
240 C% = C/256
250 POKE 71, C%             : REM ** Set up for Monitor DEC/HEX Routine ***
260 E = C% * 256
270 E% = C - E
280 POKE 70, E%
290 POKE 59, 249
300 POKE 58, 64
310 VTAB 12: HTAB 5 : PRINT "$" : VTAB 12 : HTAB 6
320 CALL 65209
330 CLEAR : VTAB 18 : PRINT "HIT ANY KEY TO CONTINUE " : GET H$ : GOTO 10
340 HOME : PRINT CHR$(12)
350 VTAB 2 : HTAB 5 : PRINT "HEXADECIMAL TO DECIMAL CONVERSION"
360 PRINT : PRINT : HTAB 3 : PRINT "Type any number from $0000 to $FFFF"
370 X = 5 : VTAB 7 : HTAB 4 : PRINT "$"
380 VTAB 7 : HTAB X : GET F$
390 IF F$ = "$" GOTO 370
395 IF F$ = CHR$(8) THEN X = X -1: Y = Y -1 : GOTO 380
400 VTAB 7 : HTAB X : PRINT F$ : Y = Y+ 1
410 F = ASC(F$) : IF F = 13 GOTO 520
420 IF F > 90 THEN F = F - 32
430 IF F < 48 GOTO 470
440 IF F < 65 AND F > 58 GOTO 470
450 IF F > 70 GOTO 470
460 GOTO 480
470 PRINT CHR$(7) : CLEAR : GOTO 340
480 IF F < 58 THEN F = F - 48
```

```
490  IF F > 58 THEN F = F - 55
500  F(Y) = F
510  X = X + 1 : GOTO 380
520  ON Y -1 GOTO 530, 540, 550, 560
530  G = F(1) : GOTO 570
540  G = F(2) + (16 * F(1)) : GOTO 570
550  G = F(3) + (16 * F(2)) + (256 * F(1)) : GOTO 570
560  G = F(4) + (16 * F(3)) + (256 * F(2)) + (4096 * F(1))
570  VTAB 15 : PRINT G
580  VTAB 19 : PRINT "HIT ANY KEY TO CONTINUE"
590  GET H$ : CLEAR : GOTO 10
```

Listing 21.1 Hex/Dec and Dec/Hex conversion program. You might want to take out 15 or 20 miniutes now to load and save this quick conversion program (be sure to boot DOS or ProDOS before entering it). It'll be handy while you're reading this book and while dabbling around in the Monitor, but you'll have to append it as a subroutine if you want it available while you're working in Applesoft.

The 6502 Divides Its Space Into Pages

One immediate consequence of the 6502's hexadecimal view of its address space is that each address can be considered to have two parts, a "page number" and an "offset." These are all 16 bit addresses (two to the sixteenth is 65,536) and the first eight bits of an address are in the page number byte while the second eight bits are in the offset byte.

Most of the Apple's memory is organized by these "page numbers." There are 256 pages of Apple memory (see Figure 21.1). The lowest is page 0 and the highest is page $FF (255). In each page, there are 256 bytes. For instance, in page $03 (3) the first byte's full name is $03 00 (768), the second byte is $03 01 (769), and the last byte is $03 FF (1023). In many machine language programs the tendency is to bounce around within a single page so only the offset byte will have to be changed.

Exploring the Landmarks in the Apple's Address Space

The Apple's address space is not a bland and homogenous box. Although the 6502's address buffer makes no special distinctions, other parts of the 6502 keep track of a few special locations. Further, once the 6502 is placed in the Apple, its address space acquires a number of relatively permanent features. These features can be considered as landmarks or semi-permanent features of the space. Each will be drawn into the map as it is described.

Special Locations for the 6502

The 6502's special locations are all at the very top and at the very bottom of the space. The ones at the top are used to help the 6502 start its work when it wakes up, and the ones at the bottom are used very heavily as a sort of scratchpad for all of the 6502's work.

Some of you out there may have been long bothered by a seeming logical inconsistency in everything you've learned about how programs are fed to the 6502. The way you make the 6502 begin to execute a program is to tell it to "jump" to the starting location of that program. However, when the Apple is first turned on, how does the 6502 know where to begin? Who tells it to turn on the disk drive and start reading in bytes?

Vectors in High Addresses

The answer is that there is a "reset signal" which runs directly into the microprocessor. The 6502 responds to the reset signal by reading the contents of locations $FFFC and $FFFD and using the number there as a guide as to where to find its first program. In all but the earliest Apples, a reset signal is sent to the 6502 automatically a few moments after the machine is turned on. You cause this special jump to take place whenever you press the "CTRL-Reset" (simply "reset" in some Apples) keys.

Those two locations, $FFFC and $FFFD (65,532 and 65,533), are called the 6502's "reset vectors." A few other bytes right up there at the top are used for similar abrupt and disorienting events such as getting started again after an IRQ or BRK "interrupt" ($FFFE and $FFFF) or "non-maskable interrupt" ($FFFA and $FFFB). Interrupts and resets are discussed in more detail in Chapter 27, but all this should serve to show that it is very important to always have meaningful information in locations 65,531 to 65,535 ($FFFA to $FFFF) in any computer which uses a 6502. These are important places in the address space.

The Zero Page

The important locations at the bottom of the space have to do with the "page" concept discussed earlier. Some of the 6502's machine language instructions aren't capable of dealing with page numbers. They work only in offsets. These instructions are used heavily for work in what is called the "Zero Page." This refers to page number 0, the very first 256 bytes in the space. Addresses in the zero page are also important in machine language programs which must be very fast or very short. Using the zero page can be fast because you only have to set up the offset byte, thereby cutting out any steps to set up a page number. Fewer steps mean less time to execute and makes for slightly shorter programs.

These various features, devoted 6502 instructions, speed, and compactness, result in the 256 bytes of the zero page being very heavily used and sometimes bitterly contested for. The Monitor, DOS, Applesoft, and a few other players are constantly vying for space in the zero page and many of the worst program disasters and most mysterious bugs have to do with collisions in the zero page. Figure 21.3 gives some idea of how easy it is for such collisions to occur.

The second page in the Apple's memory is also staked out by a few special 6502 machine language instructions. These instructions always use page number 01 and then fiddle with offsets. This page is called the "Stack" because the 6502 piles special numbers into it whenever it needs a little extra storage space and then pulls the numbers back into the 6502 off the bottom of the "stack". The 6502's demands on this page are so critical that it is considered to be completely off limits to all programs. The only permissible way in is to use 6502 stack instructions.

The 6502's landmarks are marked out in Figure 21.2 with some appropriate blow ups of the critical regions. As you can see, the zero page, the stack and the special reset and interrupt vectors don't take up very much space and leave a great open expanse between page $02 and page $FF (255). A memory map of a Rockwell or a Commodore computer will also show these characteristic reserved areas.

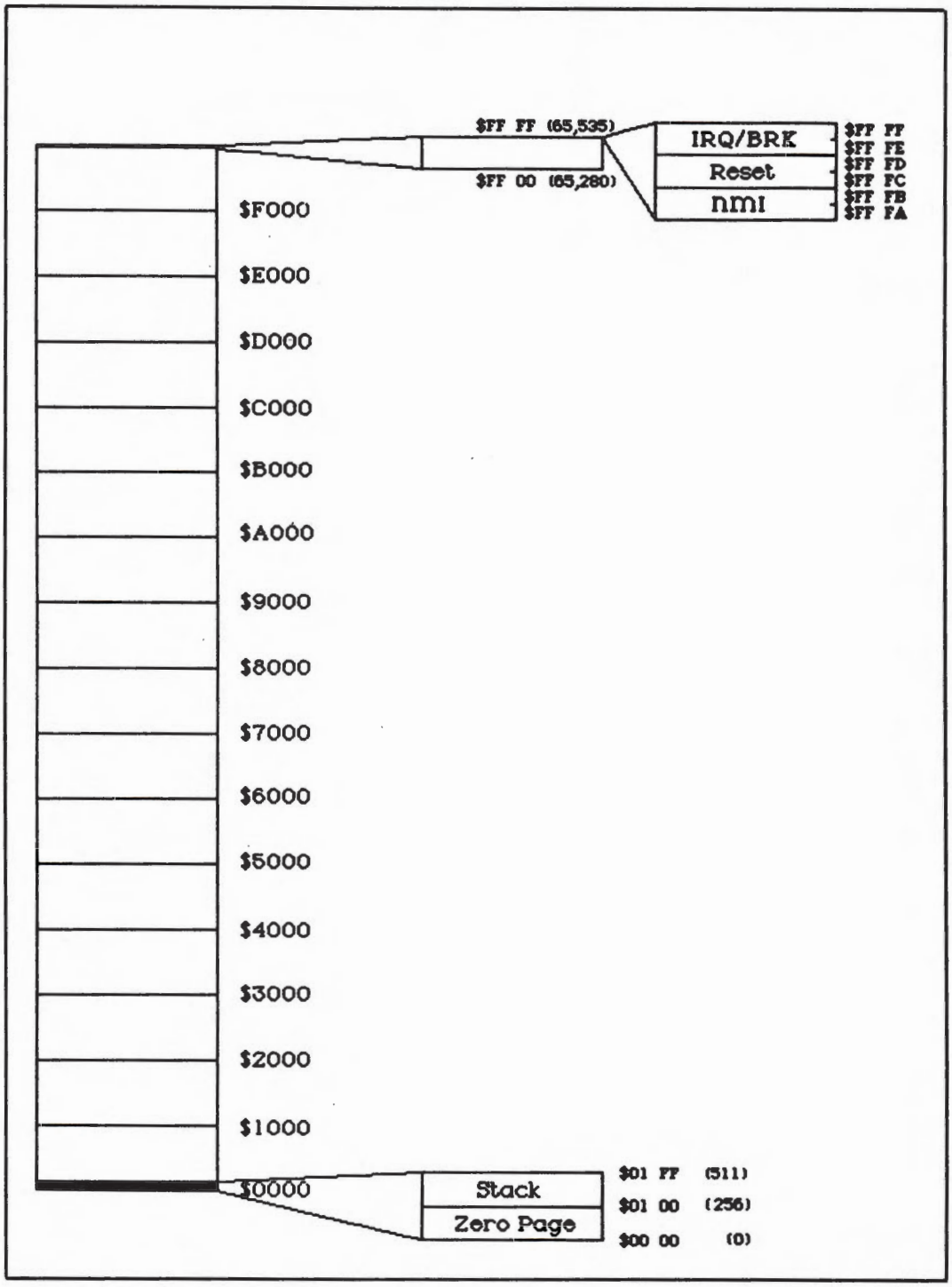


Fig. 21.2 Landmarks for the 6502. The first 512 bytes and the highest six bytes have special meanings for the microprocessor.

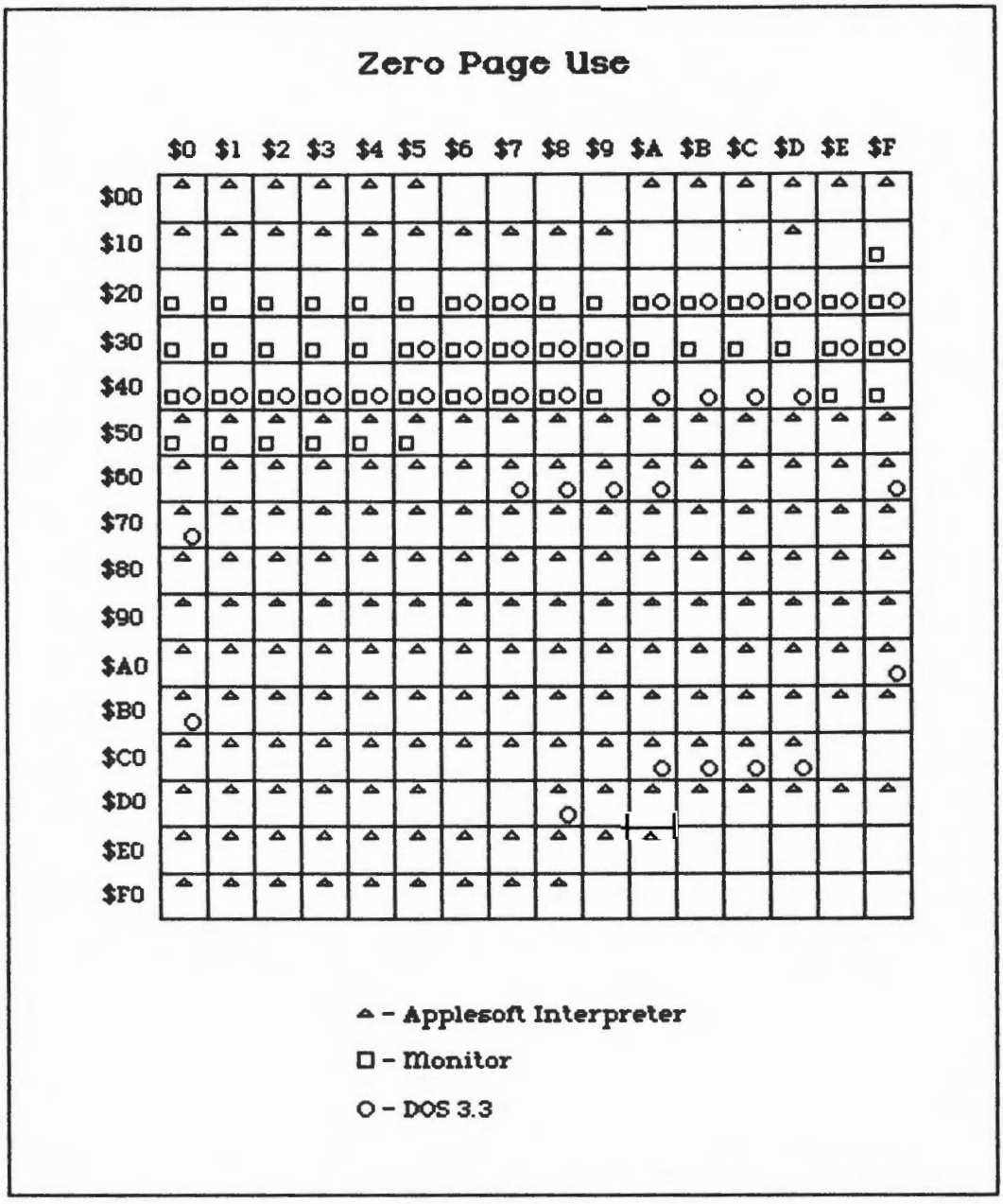


Fig. 21.3 Zero page use.

Special Places in the Apple

The landmarks in the Apple's address space fall into four general categories. Starting from the bottom, there is first a set of four pages ($00 to $04) which can be called a system scratchpad used as pigeonholes for a few important numbers. The next group is shared with the video display system and it takes up a whopping 72 pages. This is such a huge amount of memory that it hasn't been strictly reserved. Most of it can be used for other purposes if desired.

The third area includes the 16 pages of the $C000 space. The lowest page in this space, page $C0, is used for ports that help move information in and out of the Apple. It also serves as a switch plate which the 6502 can use to operate various special features of the Apple. The other 15 pages in the $C000 Space ($C1 to $CF) are usually reserved for special ROMs which hold programs for operating the cards plugged into the slots or for operating the built-in ports in the //c.

The fourth and highest area is taken up with permanently stored ROM programs including the Monitor and the Applesoft Basic Interpreter. These four special areas are marked out in Figure 21.4, and will be explored in detail shortly. The remainder can be considered free or uncommitted space available to programs to be used in any way that's useful.

Now that you're clear on the landmarks, you'll have to adjust to the fact that Apple "landmarks" sometimes disappear out of the address space. In an Apple II or II+, the entire range from 0K to 52K, including the system scratchpad, video ranges, and $C000 space, are all permanent fixtures. They are dependable landmarks which are always present and cannot be made to move. However, it is a reasonably simple task to use some bank switches to slip some RAM into the space from 52K to 64K which is normally taken up by the Monitor and Applesoft ROM.

In a //c, the $C000 space always stays put, but everything else (0 to 48K and 52K to 64K) can be replaced by "auxiliary bank switched RAM" (see Chapter 26). The //e is the most changeable of all. It can replace everything except the $C000 space with RAM, but it also has a complete alternate set of ROMs for pages $C1 through $CF.

The System Scratchpad in the Apple

Since the 6502 was going to tie up pages $00 and $01 anyway, Apple set aside a couple more pages down at the bottom for what might be collectively called a system scratchpad (shown in Figure 21.5). The whole scratchpad includes the zero page and the stack, as well as two more pages which are reserved for special uses within the Apple.

The first of the purely Apple reserved spaces is page $02 and it is called the "line buffer." No matter what is going on in the Apple, this page is supposed to be kept clear to receive characters typed in from the keyboard. As many of you know, the Apple seems to choke and gag after you've typed in 256 characters without hitting Return. The reason for the problem is that you have filled up all of the 256 available bytes in the page $02 line buffer.

The fourth and final reserved page is page $03 and it is a little different in its philosophy because most of this page has been reserved for short machine language programs written by users. At the top end of this page there are a few very critical numbers that serve to sort of connect all of the machine language software in the Apple. These bytes are called "vectors"

because each one points to the beginning of some important machine language routine. The contents of some of these vectors are listed in Figure 21.5. As you can see, some of these vectors are just about always set up by the monitor when it wakes up, but others are there only when DOS or ProDOS are loaded.

For example, one of these DOS/ProDOS vectors at location $03D0 (976) points to the "entry point" which starts running the Applesoft BASIC Interpreter. To test this out, get yourself into the Monitor (type: CALL -151 in response to the square bracket prompt, then hit Return; you should now see an asterisk prompt). Now type: 03D0G and hit return. You've just told the 6502 to check the contents of the vector at $03D0 and use it to find a program to run. If you go back into the Monitor, you will find that other nearby numbers (i.e., $03D1 or $03CF) just cause various disasters. If you try this same sequence without first loading DOS or ProDOS, the Apple will just issue a confused and distressed beep.

The page $03 vectors only take up a little bit of space, and the first couple of hundred bytes are unused. This is very important for programmers and users who need to use both Applesoft BASIC and little bits of machine language. BASIC programs and data take up huge sweeps of memory space and change in size as they run. The lower part of page $03 provides a sort of protected "safe harbor" for short machine language programs which get called from within large BASIC programs. These short machine language programs are typically loaded in beginning at $0300 (768) and then run by issuing a CALL 768 statement in the BASIC program.

Video Display Ranges

The Apple's video system is the subject of Chapters 5, 6, and 7, but the crucial idea of importance here is that the Apple has a "video display generator" which runs 14 times the speed of the 6502 as it goes about its duties of placing dots on the video screen of your CRT monitor. The video display generator works as a sort of TV camera, scanning an "image" and converting it into signals which can control a scanning electron gun in the monitor's picture tube. It is different from a TV camera in that the "image" it scans is actually a pattern in the Apple's RAM.

The 6502 goes about setting up a pattern in RAM, and the video display generator scans the pattern and translates it into signals to be sent to the monitor for display. The area in the address space in which the 6502 sets up the picture is called "video display memory." The TV camera can be pointed at any one of four different scenes, which each have their own areas in memory. These same four display memory address areas are also used as the basis for the more elaborate video systems in the //e and //c.

Text Screen One

The first of these four areas of video display memory is used so heavily in so many programs that you can just about consider it to be a reserved area, nearly as inviolate as the addresses of the zero page or line buffer. This area is called "text screen one" and it is the place that the 6502 puts letters and numbers when it wants you to see them on the screen.

Text screen one is turned on automatically as the 40 column display you look at when the Apple starts up. In the //e and //c, it is also used to provide 40 of the columns in the 80 column display. Apple owners who have a video display card in slot 3 (Videx, Smarterm, etc.) are the only ones who tend not to use text screen one since these cards have their own independent display memory (see Chapter 6 for a complete description). Another use for this range of memory is as "low resolution graphics" display memory one. When this memory is used for lo res graphics, the video display generator interprets the bytes a little differently and the 6502 puts different kinds of things into them; all of which is explained in Chapter 7.

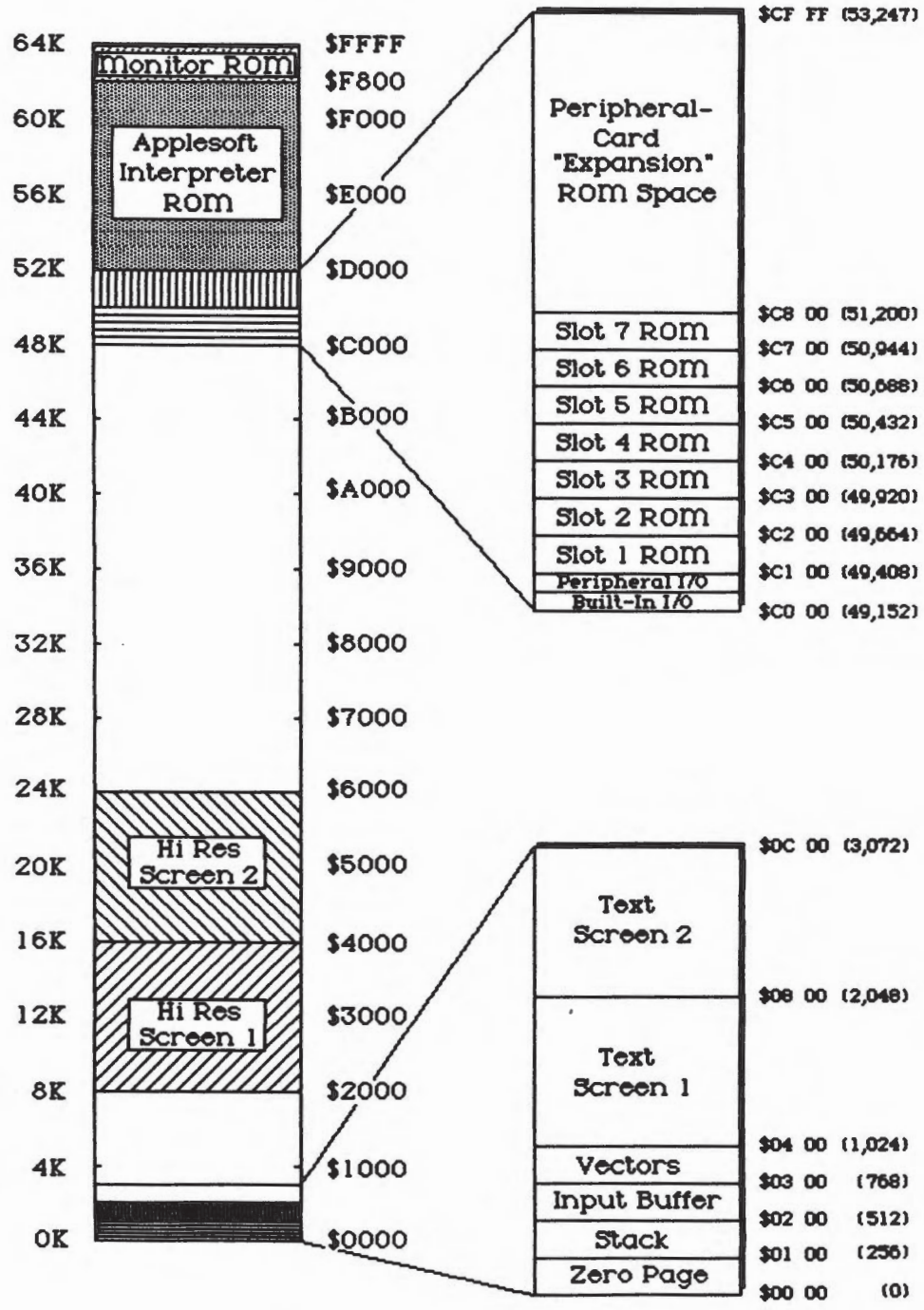


Fig. 21.4 Landmarks in the Apple II main address space.

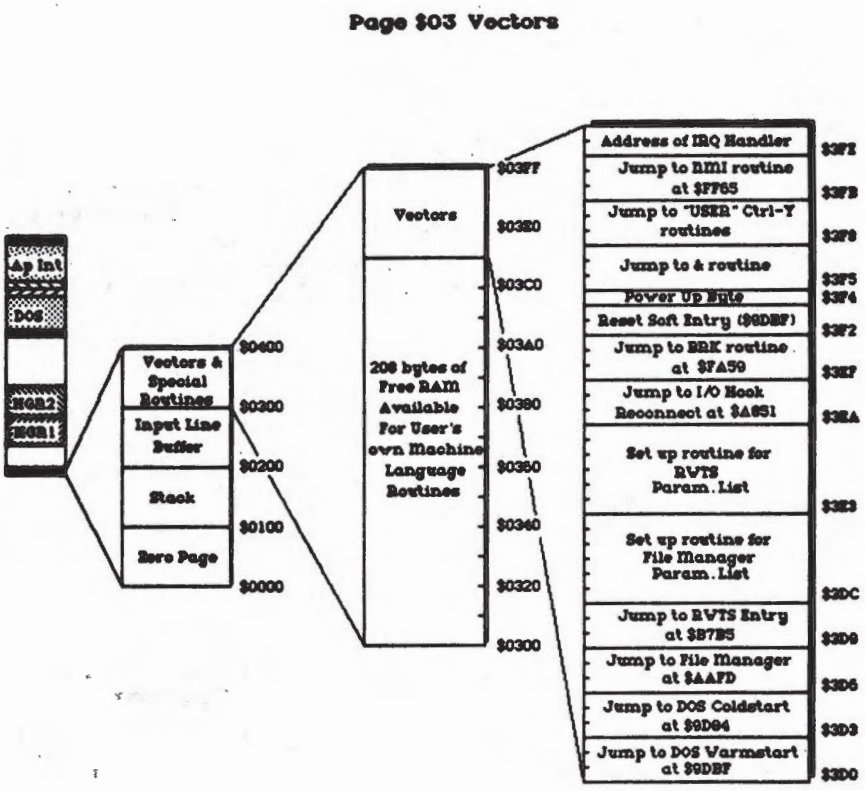


Fig. 21.6 Special "vector" locations in the $03 page with DOS 3.3 loaded.

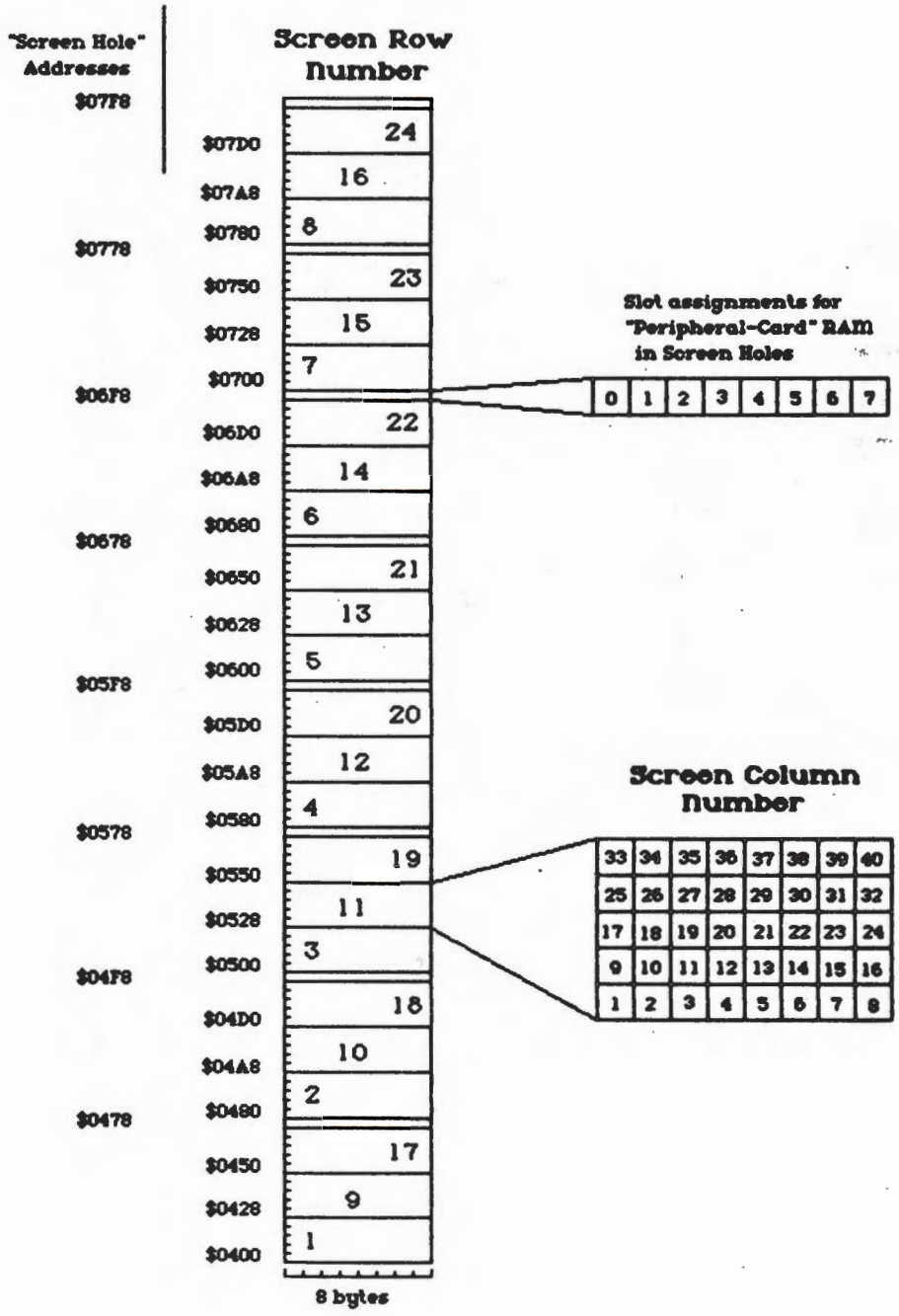


Fig. 21.6a Text screen Map. The somewhat scrambled mapping of screen memory locations to positions on the video screen is a quirk which traces back to design difficulties in the original Apple II video display generator.

Screen Hole Address Allocation and IIc Assignments

II/II+ & IIe Slot	0	1	2	3	4	5	6	7
Address Offset	8	9	A	B	C	D	E	F
Base $07Fx	Mslot $C800 owner	Ser1 Curr. Print Col.	Ser2 Curr. Line Pos.	Pascal BASH	Mouse Mode byte	Mouse		
$077x	$C0n0	Ser1 Echo Auto LF	Ser2 Echo Auto LF	Pascal BASL	Mouse IRQ Source Button Status	Mouse		
$06Fx		Ser1 Curr. Comd. Char.	Ser2 Curr. Comd. Char.	X-Coord Goto XY	Mouse	Mouse		Keyb. In Next Buf Read Pos.
$067x		Ser1 Cmd Parse State	Ser2 Cmd Parse State	In/Out Char.	Mouse	Mouse		Ser In Next Buf Read Pos.
$05Fx	Mouse Clamp Max. high byte	Ser1 Temp. Storage	Ser2 Temp. Storage	CV (cursor vert. pos.)	Mouse Y Pos. high byte	Mouse		Keyb. In Next Buf Store Pos.
$057x	Mouse Clamp Min. high byte	Ser1 print width	Ser2 line length	CH 80 Col (cursor horiz. pos)	Mouse X Pos. high byte	Mouse		Ser In Next Buf Store Pos.
$04Fx	Mouse Clamp Max. Low byte	Ser1 ACIA Status Register	Ser2 ACIA Status Register	Mode	Mouse Y Pos. low byte	Mouse		Select Port to Buffer C1/C2
$047x	Mouse Clamp Min. Low byte	Ser1 Temp. Storage	Ser2 Temp. Storage	Old CH	Mouse X Pos. low byte	Mouse		
Aux. Mem. $047x	Ser1 ACIA Control Reg.	Ser1 ACIA Comd. Reg.	Ser1 . Config. flags	Ser1 print width	Ser2 ACIA Control Reg.	Ser2 ACIA Comd. Reg.	Ser2 Config. flags	Ser2 line length

Fig. 21.6b Screen hole uses in the //c. When an interrupt occurs, the //c Interrupt Handler routine will "service" the interrupting device and then leave a flag bit set in one of the three locations marked with heavy borders in the figure. By reading these three locations, an application program can learn the source of the interrupt.

The 40 column display can have up to 24 lines of text, each with 40 letters or numbers. It therefore needs 24 × 40 = 960 bytes of display memory. Those 960 bytes of display memory are spread out among 1024 bytes in pages $04, $05, $06, and $07, as shown in Figure 21.4.

It might have been nice and simple if that area of memory was laid out as a neatly ordered set of rows and columns mapping directly to the video screen. Unfortunately, the video display generator has to do its own math at high speed with few resources. In order to make it a little easier for the display generator to figure out which byte it should be looking at on a moment by moment basis, Wozniak was forced to scramble the lines a little bit. The actual mapping of RAM locations to screen locations is laid out in Figure 21.6a.

You can have a little fun with all this by violating the space of text screen one and shoving things into it. To do this, let's first clear the screen by responding to the squre bracket prompt by typing: HOME then hitting Return. Now enter the machine language Monitor by typing: CALL -151. After hitting Return you should see the asterisk prompt. Now type: 059E: C8 C9 then hit Return.

As you can see from Figure 21.6a, $059E refers to a screen position on line 4, column 31 of the text display. What you just did was to shove "$C8" into the display RAM position for line four, column 30 and "$C9" into line four, column 32. Those two hexadecimal numbers each represent "Apple ASCII" codes (see Chapter 26, Table 26.2); one is the code for capital H, the second is the code for capital I. If you don't have DOS or ProDOS loaded, then the easiest way to get back out of the Monitor is just to respond to the asterisk prompt by typing CTRL-C, then hitting Return.

The Screen Holes

The mathematically minded among you may have been bothered by the fact that text display required just 960 bytes, but that it was spread out all the way through four pages of memory totalling 1024 bytes. That leaves 1024 − 960 = 64 bytes unaccounted for. These bytes are called "screen holes" because although they are scattered about within text screen one, nothing shows up on the video screen when you put an ASCII code into one of them.

These screen hole bytes have a special and important role. They are unique because, like the bottom part of page $03, they are guaranteed as safe protected space. However, unlike the $03 space, they are scattered about in eight small clumps of eight bytes each which makes them too fragmented to contain an actual program.

In the II/II+ and //e one byte in each of the eight clumps has been assigned for common system use, and the others are assigned to slots 1-7. There is a roughly similar assignment system in the //c (see Figure 21.6b). Since they are used by peripheral cards in the II/II+ and //e, the individual locations have no permanent assignments. In the //c; however, the parts are all built-in, so many of the locations do have assigned functions.

An alternative name for the screen holes in the II/II+ and //e is therefore"Peripheral Card RAM." These bytes ensure that every card gets eight guaranteed RAM locations on the motherboard into which it can stuff valuable numbers for temporary storage. This relieves card designers of the need to include any RAM on the card itself.

One use of the system screen holes by the peripheral cards is to keep track of which slot they are in. A card usually starts its work by running a short, built-in program to find out the number of the slot in which it has been placed. Once it has figured out which slot it is in, it writes that number into one of the screen hole RAM locations for quick reference.

TXTTAB and Text Screen 2

As you can see from Figure 21.4, the system scratchpad area, test screen one, and and the screen holes together take up the first 2K of RAM (pages $00 through $07). Most of the Apple's first 2K can pretty well be considered as reserved space where only the wizardly should tread.

The space between 2K and 48K (pages $08 through $BF), however, is pretty much open for public use. As you type in the first line of a program in Applesoft BASIC, the unanalyzed characters get dropped into the page $02 line buffer and copied into text screen one display memory. However, once you hit the Return key, the Applesoft BASIC Interpreter goes to work on your line and codes it into a series of "tokens" and numbers (see Chapter 38). That first "tokenized" line is then placed into RAM memory at the beginning of page $08 (2K). As more and more lines are entered, they are stacked one on top of another reaching steadily on upwards into free space. Similarly, when a BASIC program is read in from disk, it makes its way to the file buffer and then gets moved down in memory so that the first line begins at the start of page $08.

The usual starting point at 2K (2,048) is called "TXTTAB." Before a BASIC program is loaded you can change the setting of TXTTAB by responding to the square bracket prompt by typing: POKE 103,1: POKE 104,12 (See Chapter 40) which sets it at 3K (page $0C). Once you've done this, the next BASIC program typed in or loaded in from disk will get placed in RAM beginnning at 3K instead of at 2K. One reason you might want to do this is that the extra 1K of RAM you've just freed up can be used by the video system.

Switching Between the Text Screens

By flipping a few switches way up in the $C000 space you can "aim the TV camera" at the range of memory between 2K and 3K instead of at the range between 1K and 2K where it is aimed when the Apple is turned on. As shown in Figure 21.4, this second range of video display memory is called "text screen two" and it sits just above text screen one.

Most BASIC programmers get themselves into big trouble when they try to use text screen two. This is because it sits in exactly the same place as they've probably put their program. If you try to clear the screen in text screen two, you erase your program. For this reason and a few others, Apple hasn't included any BASIC instructions that can cause the switch and clear which could get you in trouble.

Nonetheless, for those of you who are up for a little fun with text displays, pull up to your Apple, turn it off and on, and then hit reset (CTRL-reset) to stop the disk drive. First, type in the following few lines in BASIC:

```
10 HOME
20 VTAB 5
30 PRINT "THIS IS A TEST"
```

Each time you hit Return after typing in a line, that line gets tokenized and stored beginning at $0800, which happens also to be in the range of text screen two. The line numbers and commands get turned into things which are not easily recognizable, but the letters in quotes get stored as ASCII codes.

Now we're going to hit one of the screen switches up in the $C000 range at $C055 (49,237). Type: POKE 49237,0 then hit Return. The three lines of BASIC you typed in are now displayed

in their tokenized form along the top of display screen two (notice that only the letters you put between quotes are still recognizable because everything else has been tokenized).

To switch back, you'll have to type some things which you won't be able to see. As you type them in, they will be put into text screen one for display, but the "TV camera" isn't looking there anymore. Hit the Return key just to be safe, then carefully type: POKE 49236,Ø followed by a Return. By alternately hitting either side of the text screen switch ($C055 and $C054; 49,236 and 49,237), you can toggle back and forth.

One last trick worth doing while we're at it is to tell the video display generator to treat the top part of the screen as if it were lo-res graphics instead of text. To do this, you have to hit the switch at $C050 by typing: POKE 49232,Ø followed by a Return. Just to close things out, why not take a quick check to see what screen two would look like if it also were treated as graphics. To do this, type: POKE 49237,Ø then hit Return. All frivolous stuff, but hopefully it gives you a sense of what is meant by switches and memory display screens. the various switches will be laid out a little more clearly later on in the book when the $C000 space gets explored in detail.

The High Resolution Graphics Display Ranges

The last two ranges of address space which can be scanned by the video display generator are used to set up high resolution graphics displays. The hi-res graphics display memory areas are eight times larger than the text display areas. This is necessary because the Apple must be able to individually address each dot on the screen whereas in the text mode, it is sufficient to address much larger blocks of dots with each byte of information.

When one of the text screens is being scanned, each byte in display memory is used by the video system to call up a complete character pattern accounting for seven dots in each of eight video scan lines (see Chapter 5, Figure 5.9) or 7 x 8 = 56 dots. The 24 x 40 = 960 bytes in text display memory therefore give instructions for 56 dots x 960 positions = 53,760 dots on the screen.

In high res mode, however, each byte controls only the seven dots in one single scan line (see Chapter 7). If you had just 960 of these bytes, you would only have enough information to plot out 7 dots x 960 bytes = 6,720 dots. Thus, to do the whole screen of 53,760 dots, you need eight times as many bytes. In high res, you have much more precise control over individual dots, but the price you pay is that you must have eight times as much memory set aside to lay out the entire display. Each of the high res display pages is spread out over 960 times eight equals 7,680 bytes. The first of the two hi-res display memory ranges is placed between 8K ($2000) and 16K ($4000), while the second hi-res screen is placed between 16K and 24K ($6000), all of which is laid out in Figure 21.4.

The mapping of graphics screen positions to graphics display memory locations is similar to the text mapping (see Figures 21.6a and 21.7). The program in Listing 2 will step through the whole display area and should help give you a clearer sense of how the mapping works.

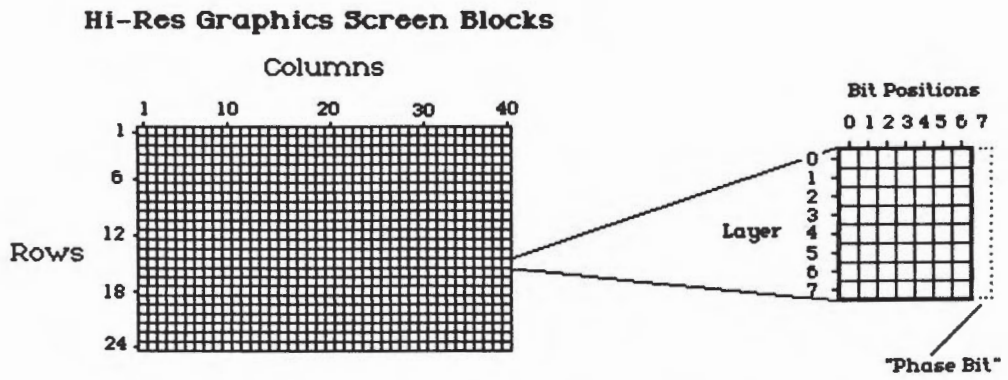
Memory Clashes Between BASIC Programs and the Hi-Res Screens

Many BASIC programs use one or the other or both of the high res graphics display ranges and so these areas are often forbidden zones for storing BASIC programs. When you switch to one of the graphics displays by issuing a HGR command in a program, any program lines which are stored within the high res display ranges get wiped out. If you use high res display one, your program has to fit between 2K and 8K. Chapter 40 is devoted in its entirety to giving you straightforward ways to break out of this memory cage as well as to providing ways which are much less straightforward.

Demo Program for Hi Res Display Memory Mapping

```
 10  HGR
 20  POKE 49234,0                    : REM ** NOMIX to see the full screen*
 30  REM
 40  FOR X = 8192 TO 16384           : REM **   $2000 through $4000    ****
 50  REM
 60  POKE X,127                      : REM **  bit pattern "0111 1111"  ****
 70  REM
 80  REM                             : REM ********************************
 90  FOR Y = 1 TO 100                : REM ** This is a delay loop so you ***
100  NEXT                            : REM **have time to watch closely***
110  REM                             : REM ********************************
120  NEXT
130  TEXT
140  END
```

Listing 21.2 Hi-res Map demo program. When run, this program graphically depicts the scrambling. You can follow the progress through the memory map, including pauses while it shoves bytes into the "screen holes."



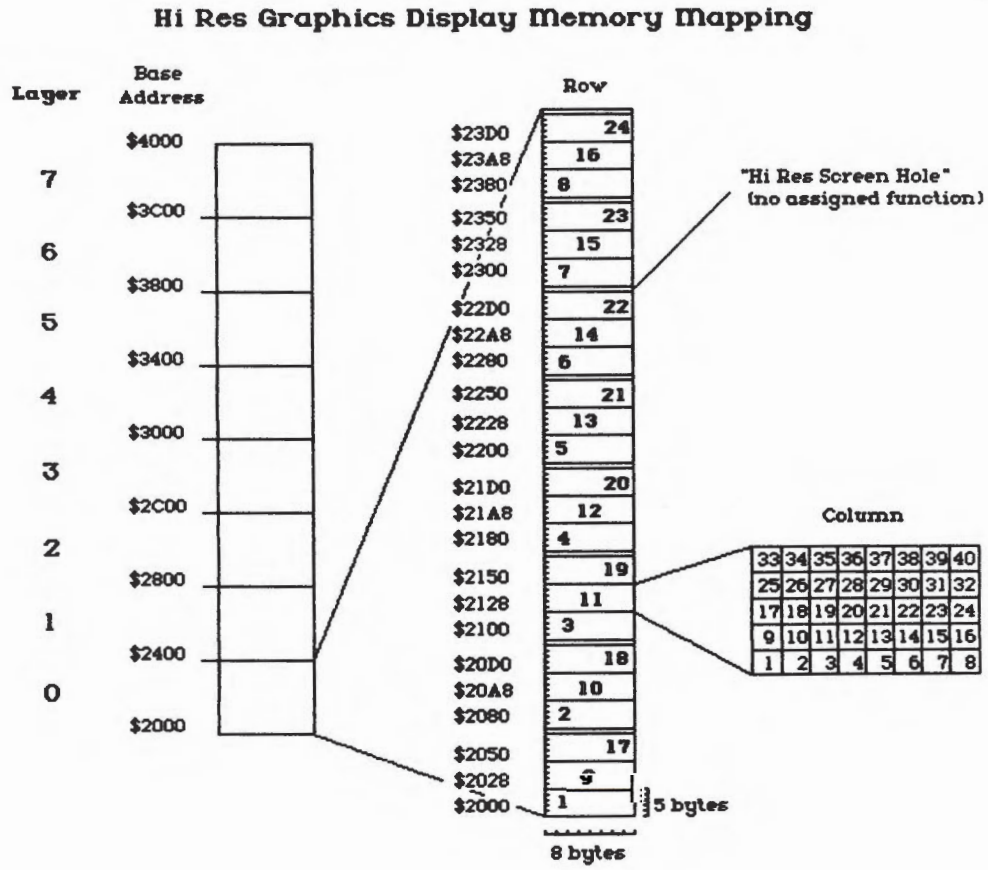


Fig. 21.7 Hi-res screen map. The layout is similar to the text screen but repeated eight times to handle the control of individual scan lines.

Filling Out the First 48K of RAM

Although just about 99.9 percent of all Apple IIs out there now have at least 48K of RAM, it wasn't always that way. You have to appreciate that in 1977 a complete row of eight 4K RAM chips cost over $100. Recall from Chapter 1 that each chip holds just one of the eight bits in a byte so you need eight of these 4K RAM chips to get 4K bytes. So, although there is room on the Apple II/II+ motherboard for three rows of eight RAM chips each, many Apples were actually shipped with just 4K of RAM installed and the remaining rows left as empty sockets.

Looking back at Figure 21.4, you can see that if those 4K bytes of RAM were placed in the bottom of the address space, you were able to have a fully functional system scratchpad, use text screen and lo-res graphics one, and still have 2K for short programs. If you decided to shell out the additional $200 for 16 more of these 4K RAM chips, you then had an additional 8K bytes of RAM, but there was some question about where was the best place to put them within the Apple's address space. One popular space was to place them up between 8K and 16K so you could play with high res graphics screen one.

When the 16K RAM chips first came out, they were very expensive, and many Apple owners could only afford to shell out $300 or $400 for eight of them. Then the question was where to put this new 16K bytes of RAM within the address space. The popular choice was to slip them in between 8K and 24K and then to move a row of the older 4K chips down to bridge the gap between 4K and 8K. This may all sound like it harkens back to some earlier dark ages, but to appreciate how fast technology moves, consider that when the IBM PC was first released in 1980, you could buy it with just 16K of RAM installed.

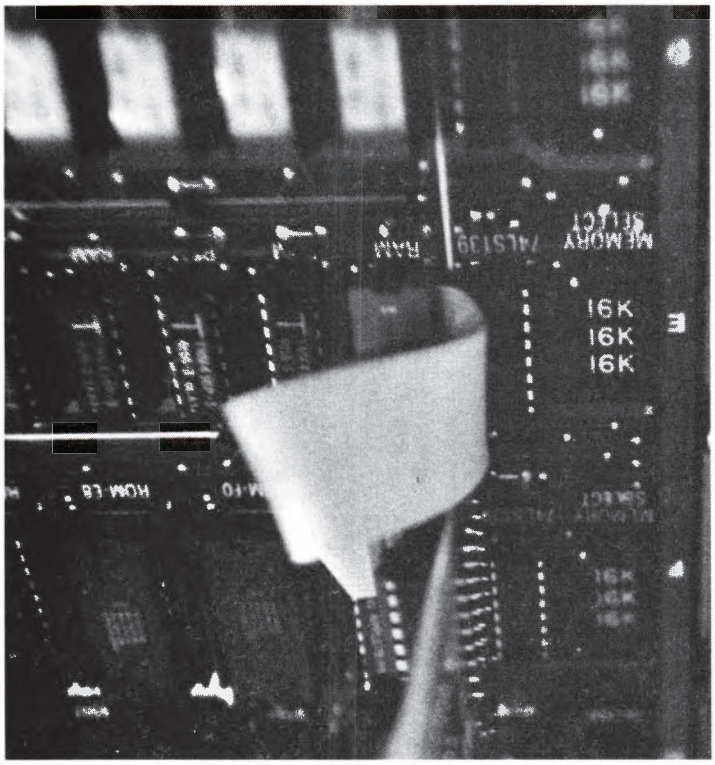


Fig. 21.8 RAM configuration blocks (here marked 16K 16K 16K) were used on old Apple II motherboards to adjust the system from the 4K of RAM on the earliest, stripped down Apple IIs, on up to a massive 48K of RAM on very expensive systems. They disappeared from the motherboard shortly before the advent of the Apple II+.

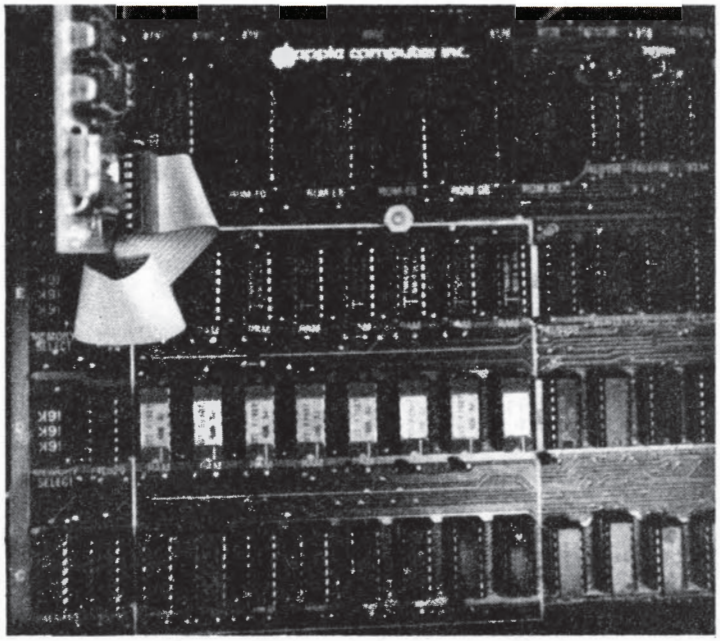


Fig. 21.9 The astute observer will notice that each of the three rows of RAM is filled with 16K chips from a different manufacturer; strong circumstantial evidence that this machine started out as a 16K Apple, grew into a 32K set up, later got boosted to 48K and finally got up to 64K of RAM with a 16K card seen in the left of the photograph.

In order to accommodate all these customized arrangements and to accommodate the mixture of 4K and 16K RAM chips, all early Apples had "RAM configuration jumper blocks (see Figures 21.8 and 21.9)." This includes all Apple II revision 0 through 6 boards. If you wanted to change the position of your RAM within the address space or if you had bought some more RAM, you pulled out the jumper blocks and rewired them.

By the time the revision 7 board was designed, all Apples were being shipped with three rows of 16K RAM chips, so the board was permanently wired for this configuration and the jumper blocks disappeared from the motherboard. Apple II+ production began shortly after the revision 7 board was designed, so only Plain Jane Apple IIs have these blocks.

One modern descendant of this system is the CramApple modification which lets you install 64K RAM chips on the Apple II or II+ motherboard so you can squeeze in 192K of RAM for a grand total of $350. Part of this sits in the regular 48K address space and the rest is used as a simulated disk drive (see Chapters 22 and 24) to feed the file buffers or is bank switched into the space. There is also a version for using 256K chips in the //e.

The Apple //e is set up for eight 64K RAM chips. In 1983, those eight 64K RAM chips cost $80. The new 256K RAM chips sold for about $900 for a set of eight in 1983, but as mass production gets underway in 1984, 256K bytes of RAM will also come to cost less than $100. In 1977, using 4K RAM Chips, that much memory would have cost over $6000 (!) and taken up nearly four square feet of board space. Legend Industries sells a card for any Apple II or III which can accommodate up to a megabyte of 256K RAM chips, the price of this board will drop steadily throughout 1984, so Legend sells its S'Card with inexpensive 64K RAM chips to be updated later when the price of 256K chips drops.

Getting Addresses into RAM Chips

One thing which might be bothering some folks is that a 4K RAM chip takes 12 bit addresses (two to the twelth equals 4,096) and a 256K RAM chip takes an 18 bit address (two to the eighteenth equals 262,144), yet all the different kinds of RAM chips are equal in size, fit into the same sockets, and have just 14 pins on their DIP packages. How do you get an 18 bit address into a 14 pin chip?. Answer: do it in two shifts, nine at a time. That means you need just nine pins for sending in addresses and you still have five pins left over to handle other details.

This uniformity is in part the product of foresight by early RAM chip designers. They set up the original 4K RAMs so that there would be plenty of extra pins 10 or 15 years later. The "pin outs" for the various kinds of RAM chips are shown in Figure 21.10. The changes in pin out with each increase in density are marked with asterisks. One important step between the 16K and the 64K variety was the simplification of the internal circuitry so that the chips only needed a single +5 volt power supply instead of three different voltages as in the Apple II's 4K and 16K RAM chips.

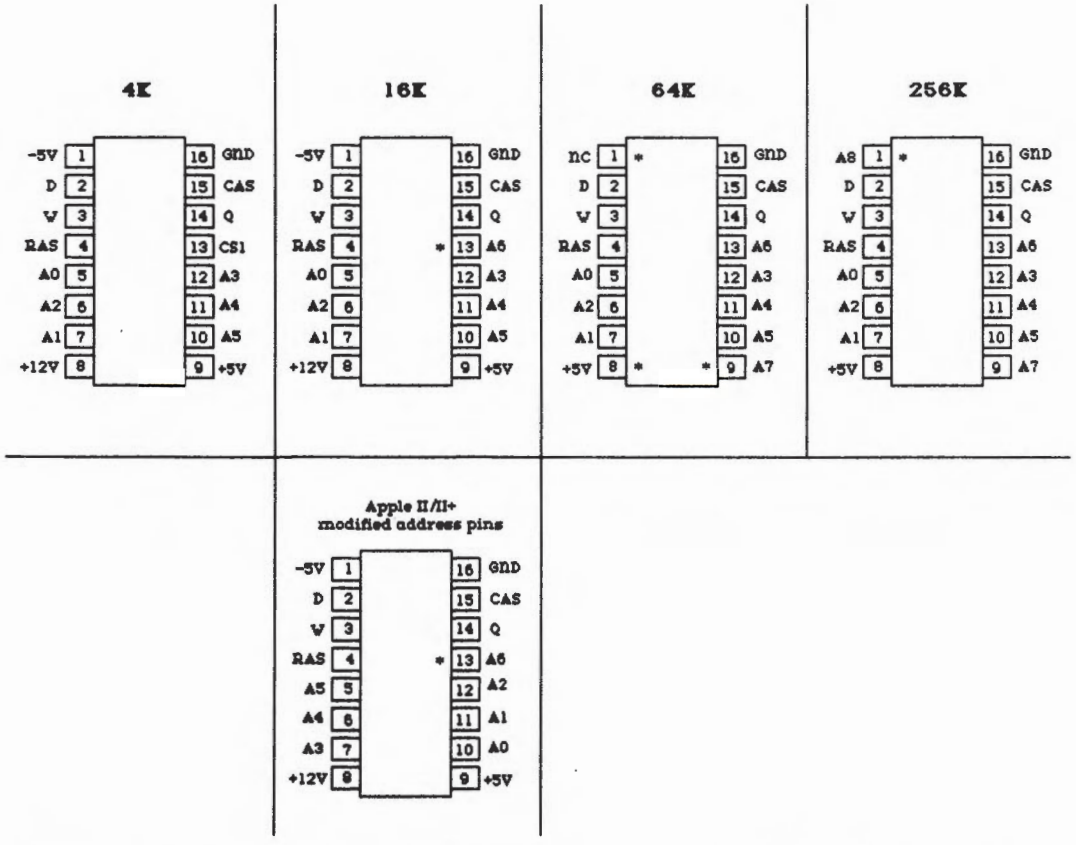


Fig. 21.10 RAM chips of increasing capacity all fit into the same 16 pin package. The asterisks mark the pin assignment changes for each new generation. D= Data in, W= Write, RAS= Row Address Strobe, CAS= Column Address Strobe, Q= data out.

The address pins are labeled "A0" through "A8". The one bit of data for each chip comes in on pin 2 labeled "D". These D pins are connected directly to the Apple's data bus. Whenever one of these chips is read, the one bit that comes out turns up on pin "Q". In Apple IIs and II+'s, the data goes first to a one byte storage buffer before getting onto the data bus; but in the //e and //c, pins Q and D are just connected together and linked to the data bus directly (see Appendix A).

The 6502 sends out a full 16 bit address, but this address always passes through several way stations before it makes it onto the RAM address pins. In the Apple II/II+, the highest two bits are used to select which of the three sets of eight RAM chips will get the addresses and/or whether the address should be directed elsewhere above the first 48K. This is one point at which the old RAM configuration jumper blocks intervened.

This leaves 14 bits of address which must be fed to the RAM chips in two groups of seven. The internal dynamic storage cells of the RAM chips can be considered to be arranged in a square matrix of 128 rows by 128 columns. There is a special clock signal in the Apple II/II+ called Address Multiplexer (AX) that operates a group of chips called the "row/column address multiplexer."

When AX is on, the "row address" of the seven bits is passed to the address pins of the RAM chips. While this address is available, the Apple sends out a Row Address Strobe (RAS) signal which causes the RAM chip to capture the row address. AX then turns off and the column address now gets sent to the pins. A few instants later, the Apple sends out a Column Address Strobe (CAS) signal which causes the RAM chip to capture the column address. Now it has all 14 bits and can use them to single out one single dynamic storage cell to be read or written to.

Most Apple II/II+ owners who have bought 16K or larger RAM cards have had to pull one of the RAM chips out of the motherboard and run a cable down from the RAM card to plug into the socket. The reason this is done is to get the RAS signal, as well as the row and column addresses off the motherboard for use on the card. Newer RAM cards use various tricks to avoid this necessity.

In the Apple //e and //c, the splitting of addresses into two eight bit parts is done by the Memory Management Unit (MMU) and it generates its own internal equivalent of the AX signal. The addresses are put onto a special multiplexed RAM Address (RA) bus. This is, in fact, the only address bus that runs up onto the //e auxiliary slot (see Appendices A and C). RAS and CAS come from a Programmed Array Logic chip which is called a PAL chip in the //e, but called TMG in the //c (see Chapter 5, Figures 5.6 and 5.8c).

Refresh

The "dynamic RAM" chips used in the II/II+ and in the //e and //c forget what they have been told to store if they are left alone for longer than two thousandths of a second. The other kind of RAM chip called "static RAM" remembers what it has been told as long as the power is on, and although static RAMs are low in power consumption and have very rapid access times, they are large and have only recently become available in 16K units. Apple uses dynamic RAMs and takes care to see that their memory is refreshed very frequently. To refresh an entire dynamic RAM chip you must address all the rows every two milliseconds, but you don't have to send in any column addresses. Whenever one row is addressed, all the cells in that row get refreshed.

Many microcomputers devote a fair amount of computer time and circuitry to managing refresh, but Apple II refreshing is one of the legendary Wozniak wonders since it requires no extra circuitry and requires no extra time.

The trick has to do with the video display generator. The display generator shares the Apple with the 6502. In each microsecond, the 6502 gets the first half, and then it has to sit there quietly while the video system takes control of the address and data buses for the second half. The video display generator is described in detail in Chapter 5, but, put simply, it has to constantly scan at least part of the Apple's memory at all times and it has to keep track of where the electron beam is in the CRT. To do all this it counts clock pulses and generates address signals.

It just so happens that out of the various counts and numbers that the video system generates, there are just enough addresses swept through to get a repeated scan through all of the RAM rows. This takes place during the video system's half of each microsecond, so it takes none of the 6502's time, and the 6502 doesn't have to generate the addresses. If you would like to read a really detailed explanation of this whole process, you should look into *The Apple II Circuit Description* by Winston Gayler (Sams Books) or *Understanding the Apple* by Jim Sather (Quality Software).

Exploring the High 16K from the Top Down

Looking back at the description of special locations for the 6502 in Figure 21.2, you will recall that any computer which uses a 6502 must have some information available in its highest bytes or else there is no way to make the 6502 start working. Some early computers had "front panels" with levers and switches to let you manually set the contents of the highest bytes before trying to start the microprocessor, but there is a much simpler and much more popular solution which is to install some ROM chips in the top of the address space. These ROMs can contain all the necessary start up programs. The Apple is designed to accept only ROM in the entire top 12K of its address space (although it is possible to switch in some RAM after the machine is turned on).

The standard layout of memory in a ROM chip is different from the layout in RAM. In the various RAM chips, the typical setup is to have just one bit of each byte stored in a chip. This is why you need eight of these "16K by one" RAM chips to get 16K bytes. ROM chips, however, are usually built with a full eight bit data output. The ROMs used in the Apple II/II+ are "2K by eight" chips. This means that although there are 16K bits of memory (like the 16K RAM chip), it is organized as 2K full bytes of ROM.

The Monitor ROM

The addresses for these 2K bytes are loaded in all at once via 11 address pins (two to the eleventh equals 2,048) and the data comes out en masse as a complete byte. Early Apples were shipped with sockets for six of these 2K ROM (see Figure 21.12) chips but in some machines only the very highest ($F8) ROM was installed. The $F8 ROM contains the Apple's Machine Language Monitor system. The Monitor ROM is mapped into the address space in Figure 21.4, and it is expanded a bit to show up some of its contents in Figure 21.13

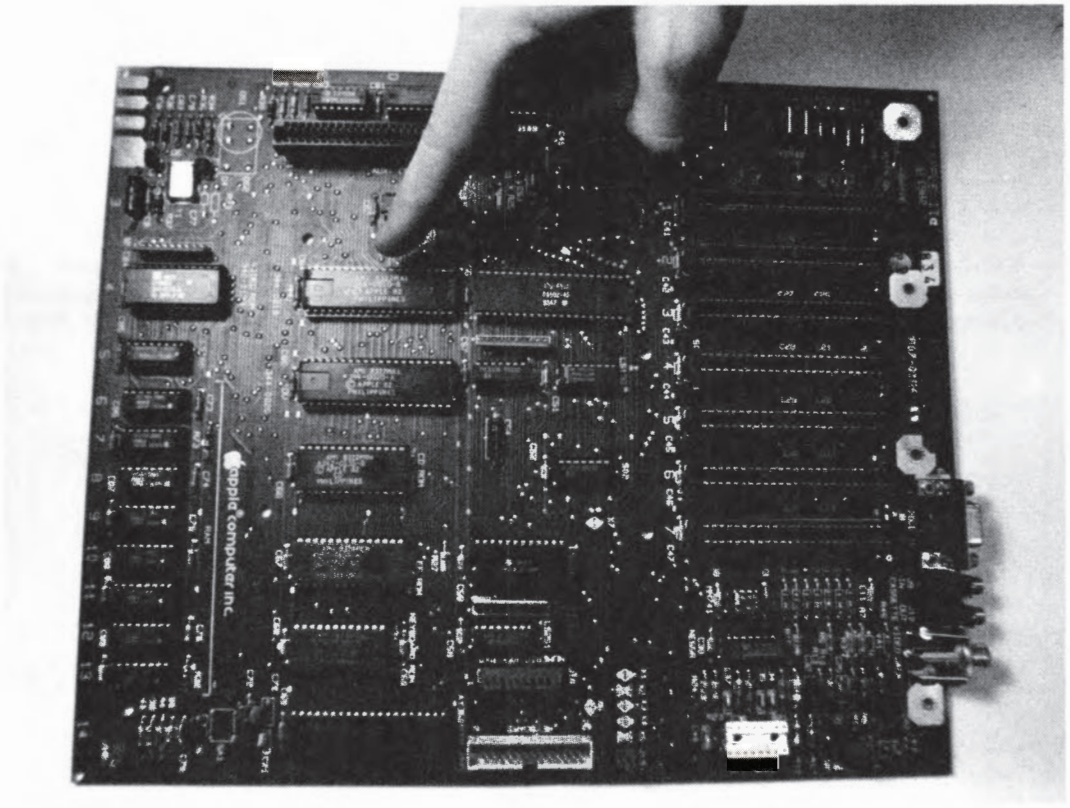


Fig. 21.11 MMU (Memory Mangement Unit).

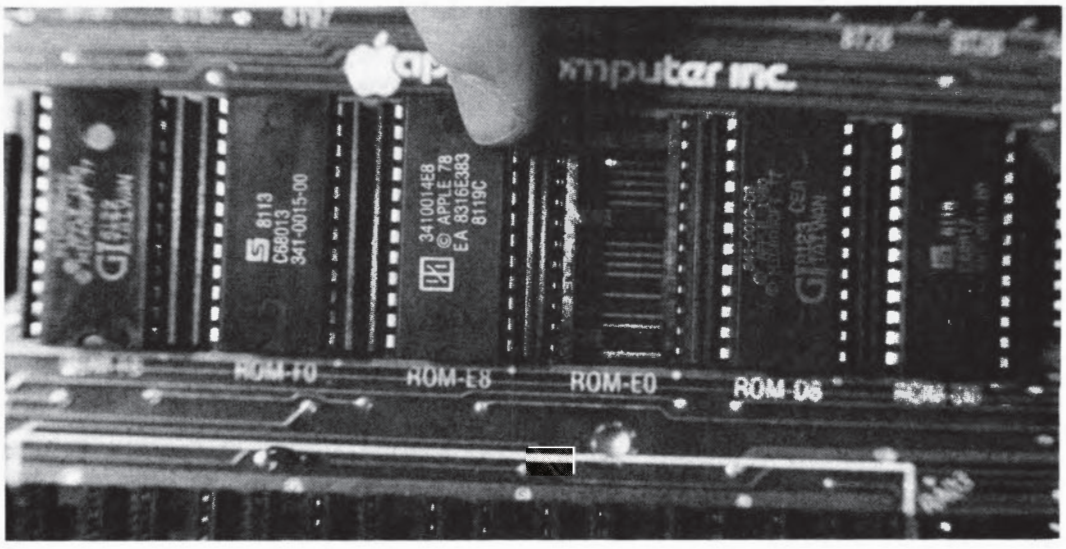


Fig. 21.12 Apple II+ Monitor and Applesoft ROMs.

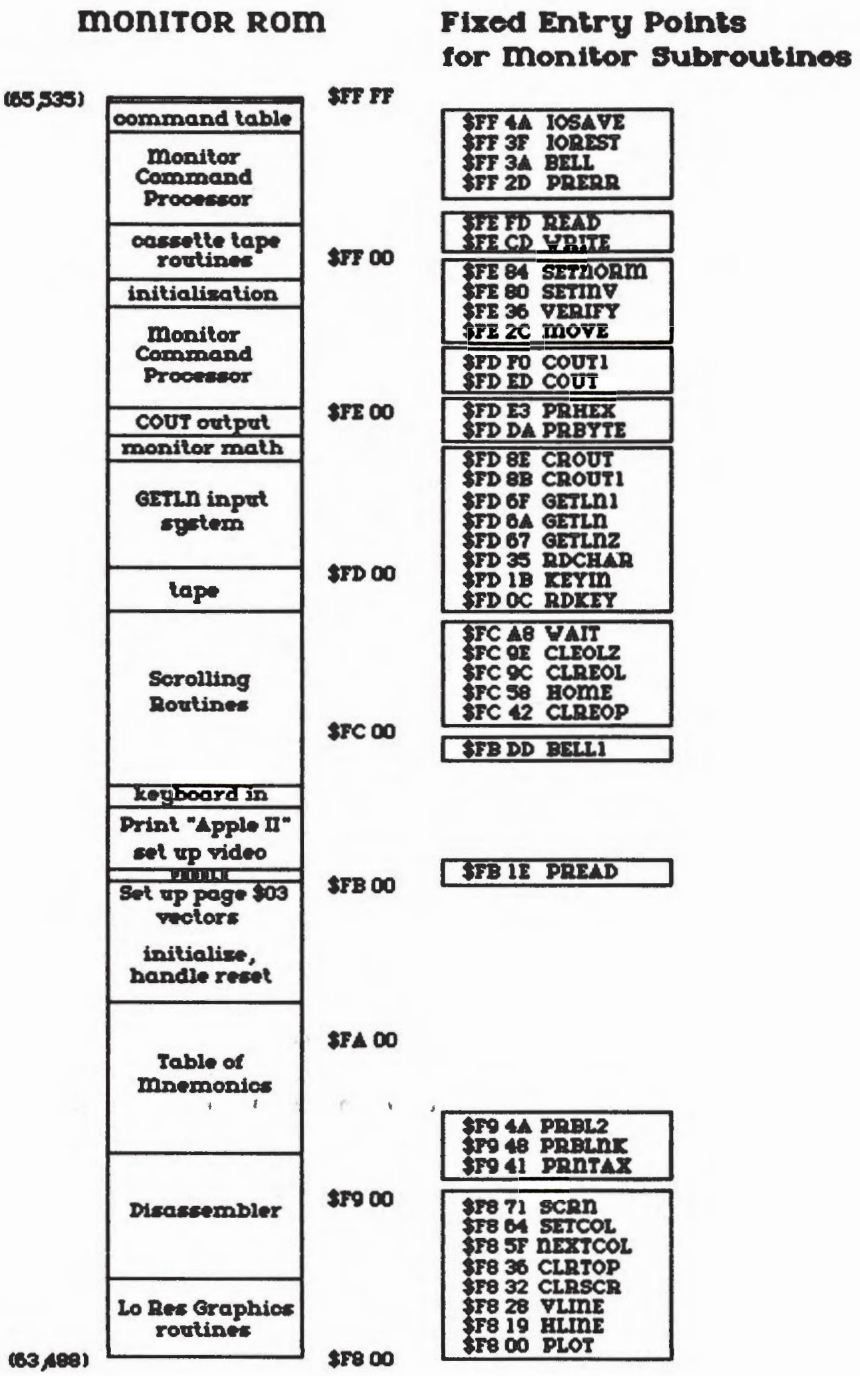


Fig. 21.13 Map of Monitor ROM and standard entry points.

The exact programs in the Monitor ROM have been changed several times since the Apple was released. All of them contain what are called "I/O subroutines" which take care of reading in characters typed at the keyboard and take care of putting the characters in their appropriate places in the text screen memory for display. The monitor also handles some features of low res graphics and is capable of reading and writing data to and from a cassette tape recorder. It also provides several mathematical and programming services for folks who write in assembly language.

There are five rather different versions of the Monitor used in various Apple IIs. The first version of the Monitor was written in 1977 by Wozniak and Baum. It is still popular among machine language programmers because it provided a variety of special features directed at that audience.

The second version of the Monitor is often called the "Autostart ROM." Putting an autostart ROM into an Apple II more or less makes it an Apple II+. With the original Monitor, the Apple just sort of sat there looking ornery when you turned it on. Early Apples also lacked the automatic reset feature, so you had to hit reset and then type some commands at the machine language level to get started. The Apple II+ was infinitely more "user friendly" in that it woke up talking BASIC and automatically tried to turn on the disk drive. Some machine language programmers who are not interested in owning a "user friendly" computer and who are not happy with the things that had to be left out of the Autostart ROM still use the original Monitor.

The original Apple //e Monitor ROM follows the II+ Autostart ROM fairly closely, but is altered in a number of places to accommodate close interaction with the 80 column firmware (see Chapter 33). The fourth version was written for the //c, and it is a major overhaul. Many of the routines have been rewritten in 65C02 machine language. These routines are faster and more compact. The cassette recorder routines have been removed, and the space that is freed up is used to expand the Monitor's "disassembler" (see Chapter 31) to accommodate the new 65C02 instructions. In addition, the input routines have been altered to accept lowercase characters in commands. The old, flawed interrupt routines have also been removed and are now handled in various parts of the $C000 ROM (see Chapter 27).

A fifth version of the Monitor has been written for the //e. It corrects the interrupt bug (see Chapter 27) and it includes a new "mini-assembler" (see Chapter 31), a feature absent since the days of the original Apple II Monitor ROM.

Despite all the shuffling there are a few routines which Apple guarantees will always start in the same place and these starting points are shown in Figure 21.13. Apple has always published full "source listings" of the Monitor ROMs so it is fairly easy for machine language programmers to find their way around. If you are interested in even more detailed descriptions of the original and Autostart Monitors, you can buy *What's Where In the Apple* by W. F. Luebbert (Micro Ink), and you can also order a disk with a database listing of all the information in the book.

The BASIC ROMs

The original version of BASIC available for the Apple is called Integer BASIC because it can only deal with numbers which are integers between +32,768 and -32,768. It was provided in ROM chips in even the earliest Apples. Integer BASIC has been almost completely replaced by Applesoft BASIC. ProDOS cannot be used with Integer BASIC.

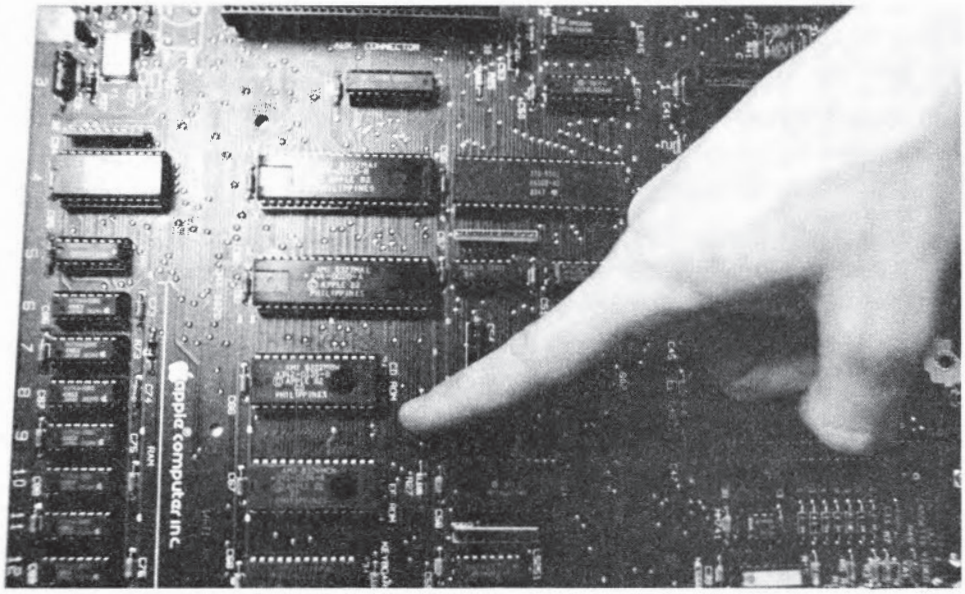


Fig. 21.14a CD and EF ROMs in the //e.

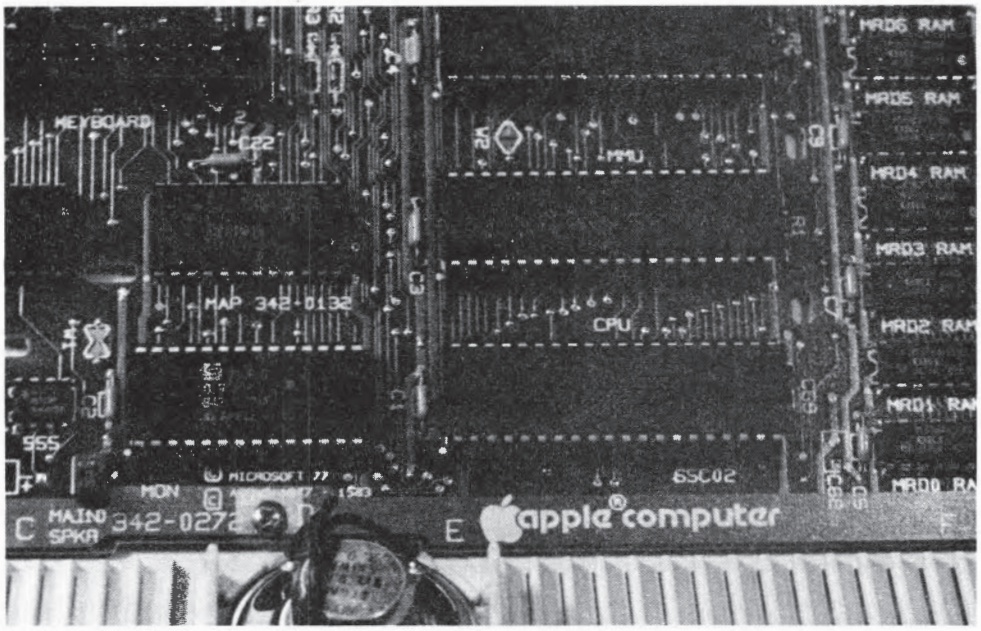


Fig. 21.14b System ROM in the //c (marked "342-0272"). The copyright marks near the ROM reflect Microsoft's role in writing the original Applesoft Interpreter. This single 16K byte ROM chip contains the Monitor, Interpreter, and I/O firmware.

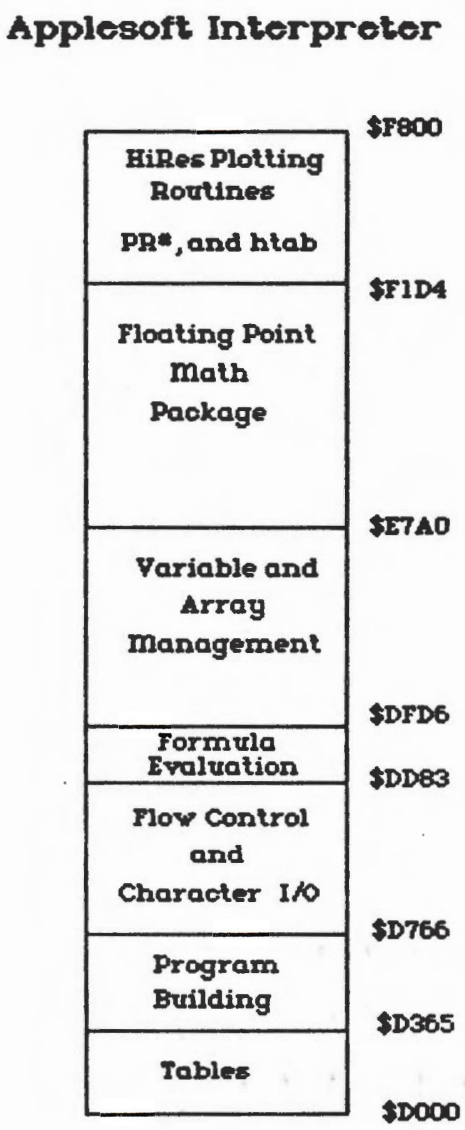


Fig. 21. 15 Map of Applesoft Interpreter ROM.

The Applesoft BASIC Interpreter is actually a machine language program which is 10K in length, so it fits neatly into the five remaining sockets for 2K ROMs on the Apple II/II+ motherboard. Figure 21.15 includes a blow up of the address space for Applesoft with some general comments on approximately what parts of the interpreter are grouped together.

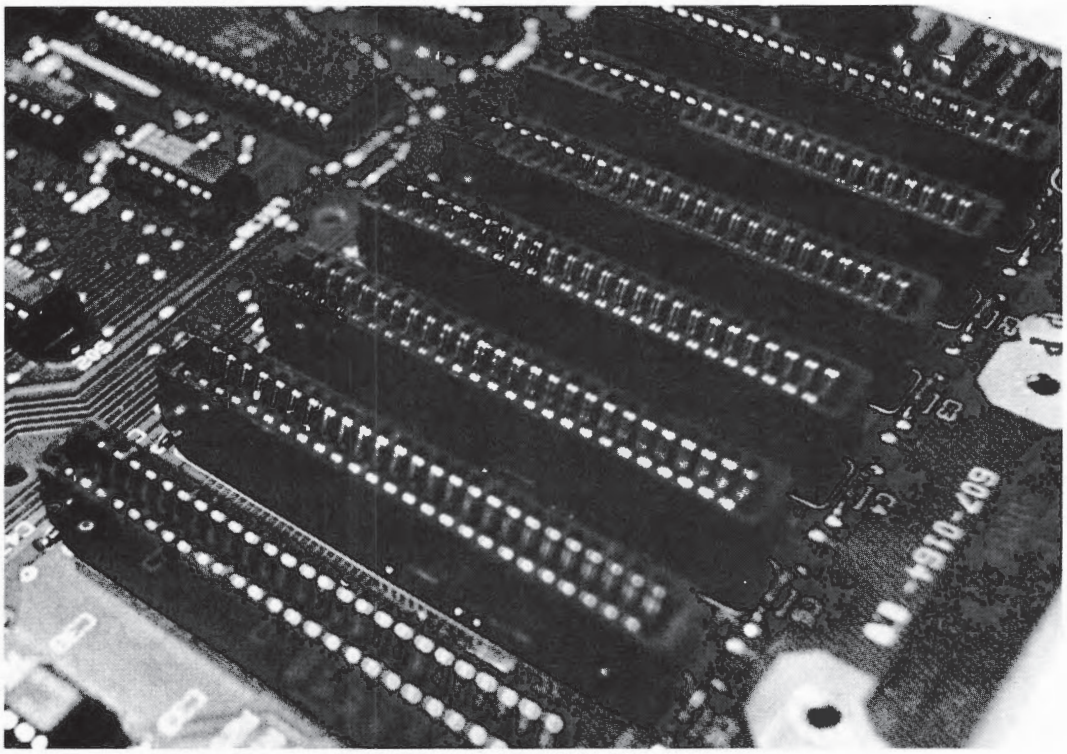
In the Apple //e, the ROMs are 8K × 8, so there are only two of them. One called the "EF ROM" includes the Monitor and most of the Applesoft BASIC Interpreter and the other, called the "CD ROM" (see Figure 21.14a), holds the remainder of the BASIC interpreter as well as some "bank select" ROM which control the 80 column text card and manages the //e's built-in diagnostics (CTRL/closed Apple/Reset).

Things are even more compact in the //c. The Monitor, the Applesoft Interpreter, and nearly 4K of additional interrupt handling and I/O routines are all packed together in a single 16K × 8 ROM (see Figure 21.14b).

You can get a complete listing of the contents of the Monitor when you buy the Technical Reference Manual for your Apple; however, you can't get a fully annotated listing of the contents of the Applesoft BASIC Interpreter (see Chapter 38). You see, the people at Apple feel very put upon by two opposing parties. One bunch is always damning them for not improving the performance of the interpreter by rewriting it every six or eight months. The other bunch has built elaborate programs which use little tiny parts of the interpreter program and these folks get upset whenever Apple changes even a single byte.

Apple has responded to all this by taking the official position that you are not allowed to know what's in the interpreter and that they therefore reserve the right to change or improve it at any time. In practice, all of the really serious commercial programmers know the contents in great detail and there have been very few updates of the Interpreter (see All About Applesoft; Call A.P.P.L.E. in Depth).

A similar situation has developed with DOS (see *Beneath Apple DOS* by Don Worth and Pieter Lechner; Quality Software), and Apple finally decided to abandon all attempts at DOS improvements and to start from scratch with ProDOS. Renewed efforts are being made to keep the internals of ProDOS officially "secret." Commercial programmers have been fully warned that ProDOS will be continually updated, improved and expanded to help improve the overall performance of the machine in the long term.



Chapter 22

I/O ROM and Ports in the $C000 Space

The $C000 Space

The 4K of address space between $C000 and $D000 is the most complex and intriguing area of the Apple. In the standard Apple II/II+ or //e this 4K range is always shipped with no RAM and no ROM in it. The //e motherboard has some ROM that can be switched into the space, but when you first turn on the Apple, this space is bare even in a //e. The $C000 space in a //c, however, is mostly filled with permanent ROM, and therefore lacks much of the mystery and intrigue of II/II+ and //e $C000 addresses.

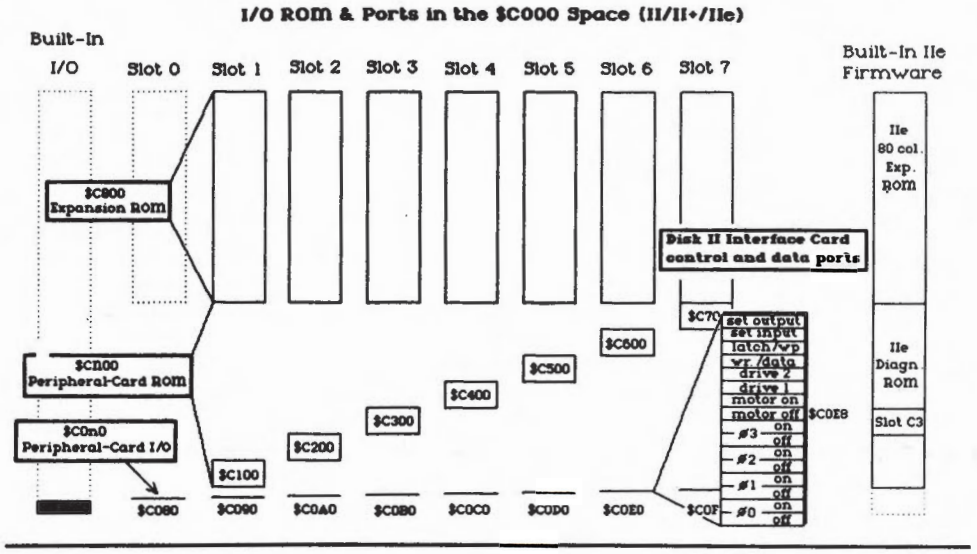
In all Apples, however, at the very bottom of this range of memory, within the 256 addresses in the $C0 page itself, there are little clumps of switch toggles, indicator signals, video controls, and a variety of paths in and out of the Apple. These switches control all of the bank switching mechanisms which can cause over a megabyte of RAM to snap in and out of the Apple's address space in no more than a few thousandths of a second. The $C0 page also provides the route through which five, 10, 20 or more megabytes of data can flow in and out of the Apple in passage between a hard disk and the Apple's file buffers.

The Expansion ROM Space

Leaving the best for last, a tour of the $C000 space can begin at the top in an empty range called the "Expansion ROM Space." This space takes up 2K of addresses in pages $C8 through $CF (200 through 207). Figure 22.1 is an expanded view of the entire 4K $C000 space. On the left it is represented in factory form, empty but for the lowest 128 addresses.

The purpose of the expansion ROM space in the II/II+ and //e is to provide some address space for larger programs stored in ROM on peripheral cards. In the //c, it contains many important system management routines (see Figure 22.1b). The addresses in the Expansion space are considered safe for use by the //c firmware for the obvious reason that there can be no peripheral cards to compete for the addresses.

Each peripheral card is allowed to have a machine language program up to 2K bytes in length. These ROMs are referred to as expansion ROMs or as "$C800 ROMs." The expansion ROM space is where these 2K $C800 ROMs along with their programs can be fit into the Apple's address space. Each of the peripheral cards is assigned the same set of addresses, but only one card at a time is allowed to actually place its 2K of ROM into the address space.



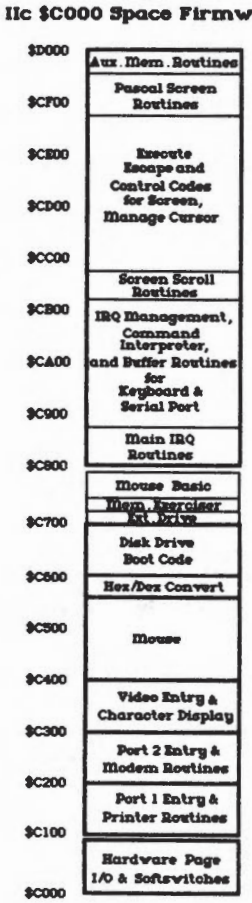


Fig. 22.1a I/O ROM and Ports in the $C000 space (II/II+ and //e).

Fig. 22.1b I/O firmware in the //c $C000 space. This single linear ROM space replaces all the complex bank switched spaces used in this address range in the II/II+ and //e.

The seven blocks drawn out in Figure 22.1a represent the situation which occurs when you have seven cards, each with a 2K ROM. Vying for position in just 2K of space is 14K of program. Slot 0 doesn't exist in the //e, and in the II/II+ it isn't allowed to have any expansion ROM.

The priority system for switching these various $C800 ROMs into the address space is based on three signals. The first of these is called "I/O Select" and it arrives at each card on pin 1 of the 50-pin peripheral card connector (see Appendix B). We'll say a little more about I/O Select when Peripheral Card ROM Space is discussed below, but you can think of this signal in some very concrete terms. When you type "PR#3" at the keyboard and then hit Return, the I/O Select line on slot 3 turns on. This is not sufficient to actually turn on the expansion ROM, but it is the key selective part of the system that controls which of the 2K ROMs will be moved into the space. The other two controls are called "I/O Strobe" and "$CFFF," and these will be described in the section on the Peripheral Card ROMs which control these signals.

Uses for Built-In Expansion ROM on Peripheral Cards

Now 2K is a very generous amount of space. Recall that the entire Apple II Monitor can be fit into just one 2K ROM chip. In fact, most peripheral cards do not need to use the expansion ROM space. As we will see shortly there is another shorter block of 256 bytes available for each card, and that is usually sufficient. Only peripheral cards which carry out fairly complex control tasks require the use of the expansion ROM space.

A good example of the kind of card which can use this 2K program space are the various 80 column cards such as the Videx Videoterm. These cards have to capture incoming keystrokes from the keyboard, place the characters into their own display memories, take care of scrolling and screen editing and even provide some graphics capabilities. Some of the coprocessor cards have large support ROMs and they are also used in some fancy clock cards to provide various easy to use services to BASIC programmers.

Although most printer cards and serial cards can get by with just the 256 bytes, several of the high performance graphics parallel printer cards such as the Microtek Dumpling GX and the Grappler+ (see Chapter 18) use $C800 expansion ROMs. This is partly because the graphics screen dump programs are fairly large, but it is also because each card is sold with several different versions of the program each set up for a different printer. The buyer looks up a table in the manual and sets some switches on the card to let the ROM know which printer it is going to be operating. The ROM can respond by running the correct program whenever it is called on to do some work.

Putting More ROM into the Expansion ROM Space

As you will see shortly, there are quite a few places in the Apple's address space into which you can stuff additional bank switched memory, and it seems that absolutely every one of these possible spaces has been staked out by at least one manufacturer. In the case of the expansion ROM space, there is one long time entrant, Mountain Computer, and a new one, Hollywood Hardware, which makes a very interesting and very useful product.

The advantage of using the expansion ROM space is that you can switch things in as good size chunks of 2K, but you don't have to worry about the tricky aspects of switching out any of the regular landmark ROM or RAM areas in the Apple address space. The disadvantage is that things can get a little complex when you're trying to use this ROM in coordination

with expansion ROM on some other card, such as a video board. The Hollywood Hardware folks claim to have this all worked out and actually offer some enhancements to the //e 80 column card's operation.

The idea of these cards is to get in behind that 2K expansion ROM space, for instance, for slot 2 and then to stack a whole series of 2K ROMs into that space. First you select that slot's expansion ROM, then you select which of several 2K ROMs you'd like to use (see Chapter 26, Figure 26.7, Item 1). For years, Mountain Computer has sold a card called the Romplus+ (see Figure 22.2) which can hold six of these 2K ROMs. The idea was that folks could put all their favorite utilities into permanent ROMs and then call them whenever needed without ever having to look for a disk to shove in the drive. Mountain was also thinking of Apples which get used in some industrial or laboratory settings doing the same fairly straightforward control and supervision tasks most of the time.

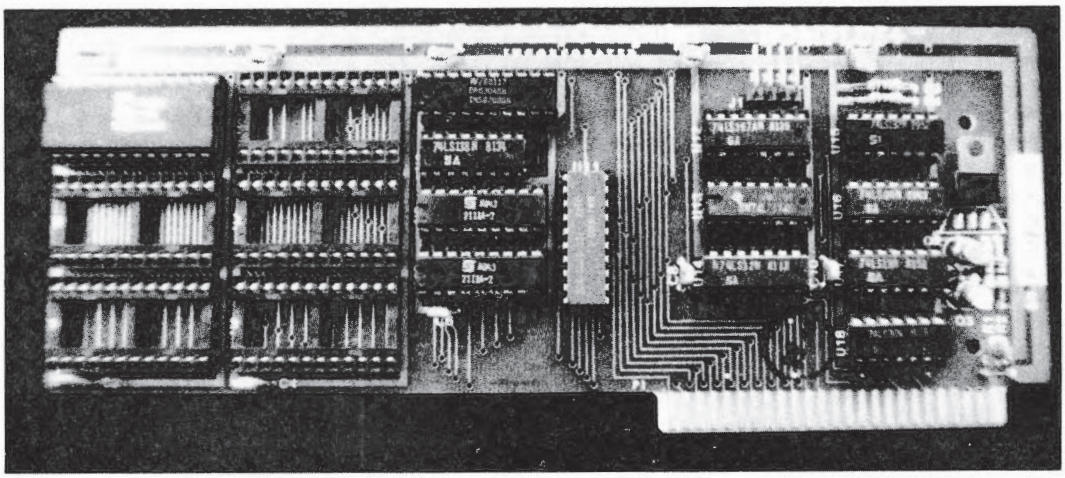


Fig. 22.2 ROMplus from Mountain Computer.

Burning your Own ROMs

For the most part, this has never really caught on very big because most folks aren't quite sure how to go about "burning" their own ROMs. Mountain Computer provides another card for this purpose called the RomWriter. It comes with fairly easy instructions for loading in a program and causing the RomWriter to make it permanent.

This works with a kind of ROM called an EPROM (for Erasable Programable Read Only Memory). These ROMs can be erased with a special kind of ultraviolet light, and then written again. Cards for writing ROMs are also sold by MPC (The AP-EP PROM It EPROM Development System) and by Hollister Microsystems (The HMS3264). The Hollister card is the most versatile since it has three different kinds of sockets and thus can burn and run a wide variety of ROMs; but it is also the most expensive, cashing in at $395.

Buying your Extra ROMs Already Burned

Actually, a ROM card with frequently used programs is a great idea, if only you didn't have to worry about making the ROMs yourself. This is what makes the UltraROM board from Hollywood Hardware such a great product (see Figure 22.3). These folks have selected a whole number of popular and available utility programs such as GPLE and a whole slew of "am-

persand utilities" for BASIC programmers (see Chapter 38) and put them into ROM for you. You can buy the card with a full 32K of programs permanently stored in ROM. The board comes with a 180 page manual and is a really excellent tool to have around if you're going to be doing much heavy duty programming in BASIC. These programs can either be used in place on the card or copied out into some other area of RAM where they are easier to get to.

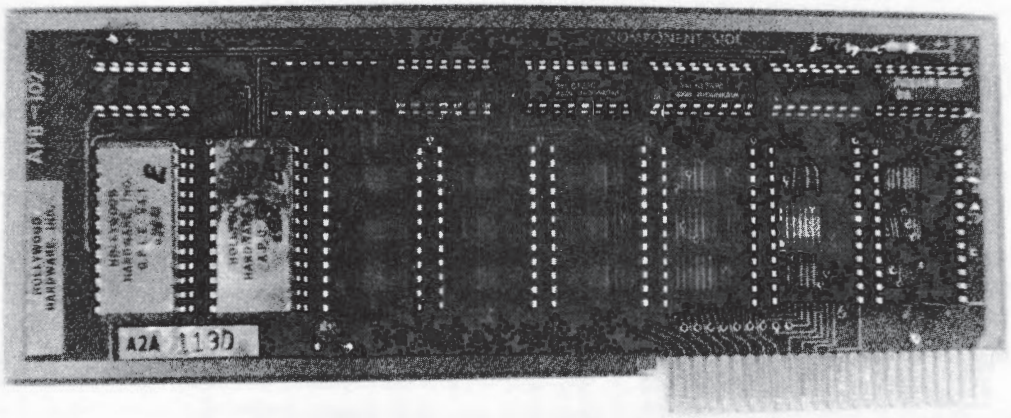


Fig. 22.3 Ultra ROM board from Hollywood Hardware.

The Peripheral Card ROM Space

Moving on down through the $C000 space you come to the "Peripheral Card ROM Space" which is made up of seven pages from $C1 through $C7. As you can see from Figure 22.1a, these pages are handled differently from those in the expansion ROM space. They are not bank switched. Each page is permanently assigned to one of the slots. They're fairly easy to keep track of since $C1 goes to slot 1, $C2 goes to slot 2, etc. (Unless of course you persist in referring to their decimal designations in which case page 193 goes to slot 1, 194 goes to slot 2, etc.).

As with the expansion ROM space, there is no ROM in the Peripheral Card ROM Space when an Apple II/II+ or //e is shipped from the factory. Once again, the //e does have some built-in ROM which can be switched into the space after the Apple is turned on.

The //c has permanently installed ROM in these spaces, but retains some semblance of the II/II+ and //e port assignment scheme (see Figure 22.1b). It is configured as if you had a printer interface in slot 1, a serial modem interface in slot 2, a video card in slot 3, a mouse in slot 4, and a disk controller in slot 6. The mouse routines spill over into the unused area for slot 5, and the slot 7 space is used for an entry for booting from the external drive, some testing routines, and for a mouse extension for the BASIC Interpreter.

Although $C800 expansion ROMs are optional on a card, these Peripheral Card ROMs (also called $CN00 ROMs because "N" changes for each slot) are all but standard equipment. In order for a card to respond to a "PR#" or an "IN#" statement in BASIC, it must have a $CN00 Peripheral Card ROM.

The principal exception to this rule is for cards which were put into slot 0 in the II/II+. As you can see from Figure 22.1a, slot 0 is not assigned a page. The $C0 page is taken up for other important functions. The one kind of card that can function well without $CN00 ROM is a "RAM card." These have traditionally been placed in slot 0 of the Apple II/II+. These RAM cards (see below for more detail) also work perfectly well in the other slots as long as you don't try to activate them with a PR# or IN# command which results in a hopelessly hung computer which must be reset or turned off. In fact, the PR#0 and IN#0 commands have nothing to do with slot 0, but are used for other purposes.

The Contents of the Peripheral Card ROMs

The $CN00 Peripheral Card ROMs have three broad functions. First, they are supposed to contain what are called "signature bytes." These are flags which let the Apple or a coprocessor test the water so to speak in order to find out what kinds of cards are installed. This will explain the surprising response you get when you select option "S" in the ProDOS main menu (see Figure 22.4). ProDOS goes out and checks each slot to see if it has a $CN00 ROM and whenever it finds a ROM, it then reads the signature byte to find out what kind of card it is. This is the basis of the report you get. Not all cards follow this signature convention, which is why ProDOS is sometimes unsure about what exactly it is you have out there.

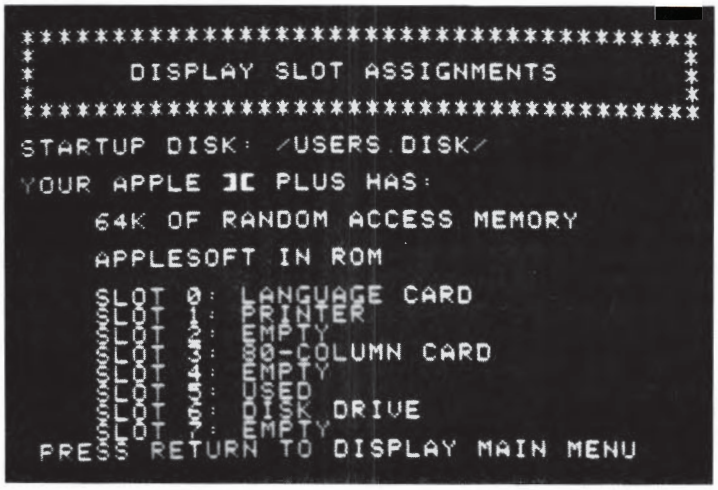


Fig. 22.4 ProDOS reads signature bytes in the $CN00 ROMs to identify cards.

These signature bytes are extremely important for disk drive controller cards. ProDOS can operate a broad variety of disk drives, but it must be able to find out from the signature bytes exactly what kind of storage media it is working with, its capacity, etc.

The $CN00 ROMs also contain short machine language programs which are used to operate the cards. An application program can read the signature bytes and decide whether it thinks it can operate the card itself or whether it should use the program stored in the $CN00 ROM as a subroutine whenever it wants the card to do something.

A card is usually "turned on" by causing the 6502 to begin executing the program stored in the $CN00 ROM. For instance if you want to boot a disk with the controller card in slot 6, you can do it by making the 6502 start executing the program at $C600. If you'd like to try this out for yourself, go over to your Apple, take any disks out of the drive, and then turn the

Apple off and on. Hit reset (CTRL-reset) to make the drive stop spinning. Now place a DOS diskette in the drive and close the door. Respond to the square bracket prompt by typing: CALL-151 and once you've hit return you should see the asterisk prompt of the Monitor. Now, to do our little exercise, type: C600G and hit Return. You can try other numbers but only C600 will work.

Address Decoding for Peripheral Card ROMs

In order for a card to figure out if the 6502 is trying to talk to it, it has to "decode" the address on the address bus. For instance, for a card in slot 2, if the address is between $C200 and $C2FF, then it must pay careful to the last eight bits of the address in order to find out which byte the 6502 is trying to read. However, for the 65,279 other addresses, it doesn't have to pay attention.

Apple figured that they could save everybody a lot of money if they relieved the peripheral cards of the responsibility of decoding the address bus. The Apple motherboard contains special circuitry which watches the address bus and then, whenever one of the $CN00 ROMs is being addressed, it turns on the I/O Select line which runs out to that card. This line is usually connected directly to the ROM chip as what is called an "enable" signal. Therefore, on most peripheral cards the address lines which pick the page aren't even connected onto the card. The ROM just watches for the I/O Select line (see Appendix B) and the address offset. Apple hoped that this would simplify the circuitry on peripheral cards and thus make them less expensive.

Phantom Cards

One trend in high performance peripheral cards is to put several functions on a single card but make the the whole thing seem to act as if each function was in a different slot. For instance, the Versacard from Prometheus has a parallel printer port for slot 1, a serial modem port for slot 2, a clock for instance for slot 7, and a few other odds and ends. All of this stuff is actually on just one card which you could put into for instance slot 4. Once it was set up, you'd type "PR#1" and your printer would turn on. A PR#2 would send characters to your modem, etc.

This little trick is called "phantom slots" and it is also used in cards from Mountain Computer (CPS) and from Videx (PSIO). The way these cards are able to do this is to decode the page address for themselves, and not rely on the I/O Select line. When the 6502 sends out $C200, the motherboard decoding circuitry tries to turn on slot 2. Meanwhile, the Versacard has been sitting there in slot 4 with its ears open, and when it picks up the $C200 signal on the address bus, it circumvents protocol and turns on its serial modem function.

The Ultraterm //e Trick

Apple was rightfully proud of its elegant and simple 80 column text system on the //e, in fact, they were so happy with it that they decided to make it impossible to run one of the older video boards in slot 3. To do this, the //e motherboard was wired so that if a card is installed in the auxiliary slot, the I/O Select line for slot 3 never gets turned on.

The //e 80 column card system does have a few limitations which some people aren't happy with. For one, it doesn't do to well with 1200 baud modems (see Chapters 6 and 17) and for another it generates a five by seven dot character matrix while many II/II+ users had grown accustomed to nine by 11 dot or at least seven by nine characters. However, the Apple //e reference manual says on page 133: "Installing an 80-column text card in the auxiliary slot

makes it impossible to run any peripheral card that has built-in firmware in slot 3." This is a problem if you want both a 64K card in the auxiliary slot for super high res graphics and a high resolution, "slot 3" 80 column card.

How many of you have guessed what the folks at Videx did for their new Ultraterm card? That's right, they ignore I/O Select and decode the bus themselves. Immediately upon waking up the Ultraterm swats out the //e 80 column firmware and then goes about its business unperturbed. Besides being able to use an Ultraterm and still have something in the auxiliary slot, there is an added bonus. The Apple's high res video display is completely independent of the whole slot card I/O system. It is therefore still possible to use extra RAM in the auxiliary slot to display super high res graphics at the same time that the Ultraterm is displaying text. To do this you need two video monitors, one connected to the Apple's regular video port for the graphics and the other connected directly to the Ultraterm for text (see Figure 22.5).

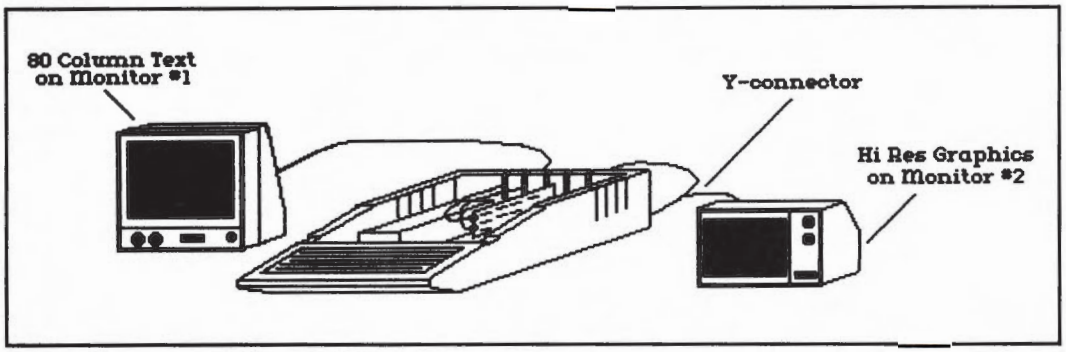


Fig. 22.5 With any 80 column card in slot 3, you can see hi-res graphics on one monitor simultaneous with text on a second monitor. This is very helpful when programming full screen graphics. If you have a card in the auxiliary slot of the //e, however, this only works with an Ultraterm board.

Using the $CN00 ROM to Control the $C800 ROM

The system for turning on the $C800 2K expansion ROMs is a two tiered process. Usually, the expansion ROMs are turned on by the $CN00 programs. If you will recall, the I/O Select line is also used as part of the enable system for the bank switched $C800 ROMs. However, the Apple motherboard provides some additional decoding for the expansion ROMs. Whenever someone, usually one of the $CN00 ROMs, refers to any of the addresses in the $C800 space, a signal called "I/O Strobe" on pin 20 of the expansion bus is turned on at all of the slots. For an expansion ROM to be turned on, it must have both an I/O Select signal (which tells it that its slot is being used) and an I/O Strobe signal (which tells it that an address in the expansion space is being called).

The third and final element of the system is a switch for turning off all of the expansion ROMs. This is done simply by referring to address $CFFF, the highest address in the $C000 space. All expansion ROMs are supposed to respond to this signal by turning off.

The whole sequence for using a program from an expansion ROM goes as follows. First, someone turns on one of the peripheral cards by causing the I/O Select line to be turned on for that card, i.e., with a PR# command. If the peripheral card ROM on that card wishes to

use its own $C800 expansion ROM, it first hits the $CFFF switch which forces any active $C800 ROM out of the address space. It follows up by calling any address in the range of $C800 to $CFFE and this makes the I/O Strobe signal turn on. The $C800 address space is now clear, and the combination of I/O Select and I/O Strobe then switches the expansion ROM on the active peripheral card into the $C800 address space (see Appendix B).

Making the Peripheral Card ROM Space Work Even Harder

As mentioned earlier, if there is a nook or a cranny in the Apple's address space, someone will have built a card to shove extra memory into it, and the Peripheral Card ROM Space is no exception. The participants in this particular derby include Axlon and Microtek, and we're now starting to talk about really serious amounts of memory. The Axlon systems include a 320K external box (see Figure 22.6) that physically looks like a disk drive and a more conventional looking 128K card, and there are two cards from Microtek; the BAM 128 and the Q-Disc.

Fig. 22.6 The Axlon RAMdisk 320 to the left of the Apple is about the same size as a standard disk drive.

All of these systems use an on-card bank switching system to move little 256 byte chunks of RAM into a sort of window provided by the Peripheral Card ($CN00) ROM space assigned to the card (see Chapter 6, Figure 6.7, Item 2). In the 128K systems there are 512 of these 256 byte pages and they are selected by putting a number between 0 and 255 into a control port (explained later) and either a 0 or a 1 into a second control port to select which set of 256 pages should be used. The difference between the two Microtek cards is that the BAM 128 can only be used for storing data from BASIC programs and for running large VisiCalc models, while the Q-disc has got added hardware and firmware which lets it simulate a "plug and run" extra disk drive.

These are just a few of the numerous "RAM disk" systems, the rest of which will be discussed later. The term RAM disk means that data is stored in and retrieved from RAM instead of from a disk drive, but that the data is brought into the computer via the DOS file buffer as if it were coming from the disk. This means that it never has to be in the Apple's address space until it is ready to get moved into the file buffer.

This is a nice approach to the problem of how to get access to all that memory, because when it is time to move some data into the file buffer, it requires absolutely no address space other than the 256 byte peripheral card ROM space which is already permanently assigned to it. Most of the other systems are just marginally faster because they use bank switching of very large chunks of RAM, but the larger the area that gets switched, the greater the potential for disruption of some other vital function in the machine. Yet other "RAM disks" do no bank switching at all but instead feed all the data through a single "port" one byte at a time, which is a comparatively slow approach. These various systems will all be compared when all of them have been described in more detail.

One important thing to remember about these systems is that they operate very differently from the most other bank switched type RAM disks or RAM expansion cards which are based on the high 16K of the address space (see Chapter 25) or on the //e auxiliary RAM (see Chapter 26). RAM expansion software written for the other systems cannot readily be modified to work with these systems.

The Axlon 320K Ramdisk is a particularly nice package if you're prepared to spend a large sum of money on a high performance system. A typical application where this much RAM comes in handy is the use of one of the large spelling checker programs. There is enough space to load the proofreading program, its dictionary, and your text all onto a single RAM disk, so that all of the activities in the proofreading process benefit from the high speed.

You can get nearly this much RAM with the Synetix card (see Chapter 24), but Axlon has three advantages. First, it uses an external power supply built into its box; second, it has little flashing lights that indicate when the drive is in use; lastly, it has battery backup. You don't realize how much you depend on the whirring noises of a drive to confirm activity until one day you find yourself wishing you had auditory confirmation that your RAM disk actually was engaged in that dBase sort. The disadvantage of the Axlon box relative to the more expensive Pion box (see Chapter 24) is that the Pion system can be upgraded to one megabyte.

The Mountain Expansion Chassis

If all your slots are full, and you feel you absolutely must have just a few more cards, you can sometimes solve your problem by adding eight additional slots in the form of a Mountain Computer Expansion Chassis (see Figure 22.7). However, there are a number of limitations on how this device can be used. And it's expensive, so you should make certain beforehand that it's appropriate to your problem.

There is absolutely no way that the Apple can be made to address more than eight slots; however, it is possible to apply the concept of bank switching to the actual physical slots themselves. This is the principal behind the Expansion Chassis. You don't actually get to work with 16 ordinary slots, rather you have a choice of which set of eight will be active at a given time.

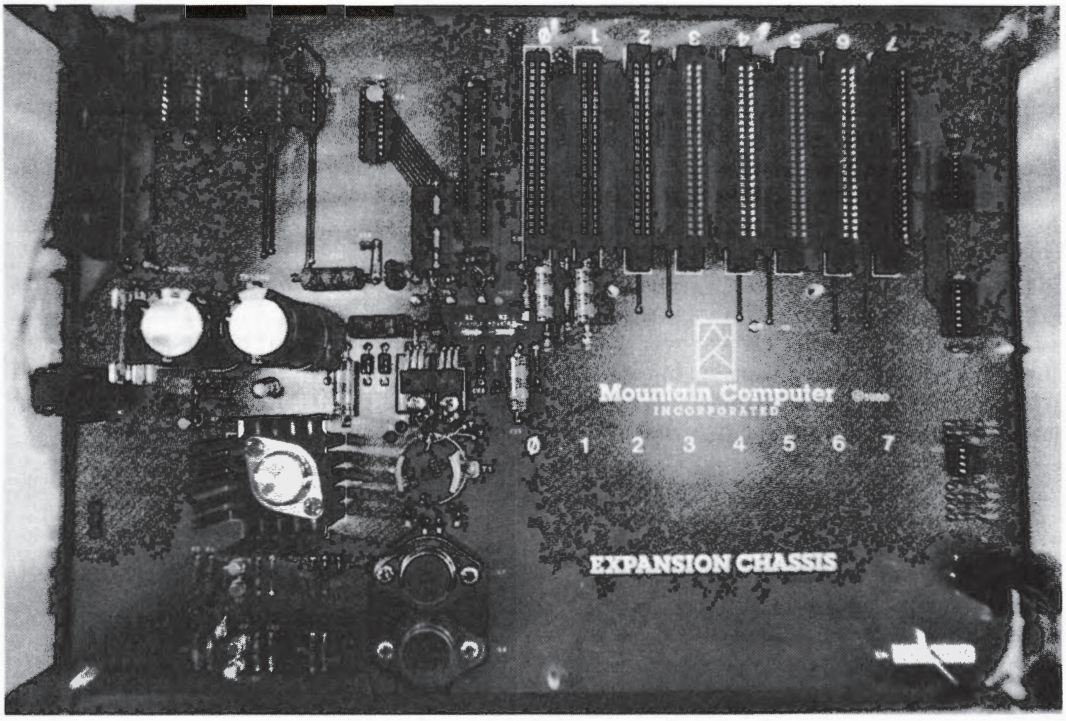


Fig. 22.7 Inside the Mountain Expansion Chassis.

When you install the Expansion Chassis you pull a crucial chip out of the Apple II/II+ motherboard (the LS138 at F12 for those who count) and plug a cable from the chassis' interface card into the empty socket. This connection gives the interface card control over the I/O Select and Device Select lines. When properly instructed, the interface will make sure that none of the slots in the Apple receive a select signal when they are addressed, and it will generate its own select signals to send out to the new slots in the external chassis.

Mountain has chosen to use the address of the cassette port ($C020) as an operating switch. However, this is not a simple one bit on/off arrangement, because you can select any one of four different chassis in a single Apple, depending on what value you write to $C020. One nice feature is that you don't have to throw all eight slots at once. You can set switches on the interface card which will effectively lock selected slots into the Apple. For instance, if you have your disk drive controller in the Apple's slot 6, you might lock that slot into the Apple. Otherwise, when you were working in the chassis, you would have no access to the disk drives. The bad side of this is that until you turn the machine off and alter the switch, the chassis' sixth slot is out of commission.

This may sound great so far, but there are three crucial flies in the ointment. The first is just a fault in the electrical design. There is enough stray capacitance (see Chapter 13) and related problems that the connection between the chassis and the Apple introduces a "propagation delay" for signals passing between the two. For most cards, this is no problem; however, for such coprocessor cards as the popular Microsoft SoftCard, this delay overwhelms the extremely precise timing demands for complex control of the bus (see Chapter 24). A second problem is mechanical; the very longest cards, such as the Vista A800, do not physically fit into the chassis.

The third problem is a little more subtle but it is the one which imposes the most severe limitations. When you throw control out to the chassis, you also throw the $C800 space out there. Unfortunately, if you were using an 80 column card, your video display ROM is now swinging in the wind, cut off from the 6502. Next time the Apple tries to send a character to the screen, blammo, you have a hung Apple. To avoid this, you must carefully turn off the video with a PR#0 (we're talking II/II+ here) and then hit $C020 to go out to the chassis. While you're out there, you can't send anything to the screen. When you finish your work in the chassis, you can come back up into the Apple, reactivate the video board and then continue.

To partially solve this problem, Mountain lets you lock the $C800 space to the Apple. If you do this, however, then you can't use any cards with expansion ROM in the chassis. What you really need is slot by slot control over whether the Apple or the chassis has the $C800 space. There is a way of rigging that up, but it's not included with the chassis. An alternate solution in many cases is to buy a second video board just for the chassis. This leads to some interesting possibilities for multiple displays, but it also adds to the expense.

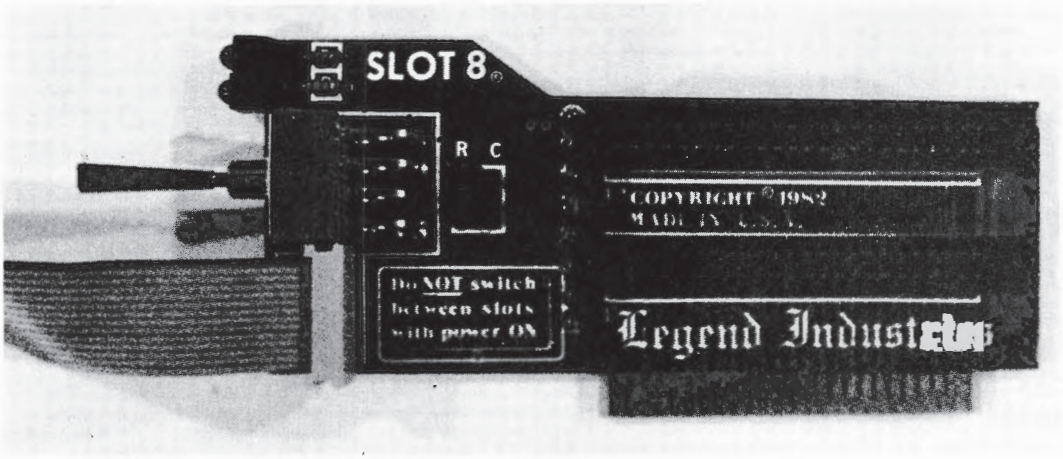


Fig. 22.8 Slot 8 from Legend. This unit plugs into slot 7 and holds two cards which extend horizontally. Only one is active at a time, and you must turn off the Apple to switch between them.

In summary, if you have a fairly good conception of what's going on out there, you can program and work your way around most of the problems. If you are using simple interfaces such as A/D converters, timers, "dumb parallel ports," etc., then the expansion chassis provides a very handy means of putting a large number of device interfaces under the control of a single Apple. Smarter cards with expansion ROMs cause a bit more trouble and very smart cards, such as the Microsoft Z-80, just won't work. A much more limited slot expansion system that does work with any card is shown in Figure 22.8.

The $C0 Page

The tour has now progressed down into the deepest level of the $C000 space, the $C0 page itself. As you can see from Figure 22.1a, this page represents just one small 256 byte part of the total $C000 space, but as you can also see if you peek ahead at Chapter 26, Figure 26.1, it has sure got one awful lot of densely packed chunks of Apple landmarks.

There are two fairly distinct parts of the $C0 page. The highest 128 addresses are collectively called the "Peripheral Card I/O Space" while the lowest 128 bytes are called the "Built-In I/O Space." In the //c, everything comes built-in, so there is no obvious distinction between the top and bottom half. Further, some features in the page are not really related to I/O. Thus, the entire $C0 page in the //c can be more appropriately referred to as the "Hardware page."

Throughout the $C0 page, you will find a resounding absence of RAM and ROM. Instead, this page is filled with such things as switches, toggles, strobes and latches. These are made up of various kinds of circuit components which can behave rather differently from RAM or ROM chips. Many of the addresses can pass only one bit of information rather than a whole byte. Some of them can be written to but can't be read. Many of the locations aren't even connected to the data bus and some never get any real contact with the address bus.

In the Built-In I/0 Space, it is possible to give an exact description of every important part. The Peripheral Card I/O Space, on the other hand, contains whatever happens to be on the cards that get shoved into it.

For the most part, however, throughout the $C0 page there are three broad categories into which the residents of the addresses fall. The greatest number can be called "control ports," through which the 6502 is able to modify the behavior of devices outside the address space. There are also a fair number of "status ports" which permit the 6502 to get readable reports about the behavior of these devices, and finally there are "data ports," the actual locations through which individual bytes or bits enter or leave the address space.

The Peripheral Card I/O Space

In the top half of the $C0 page, each slot, including slot 0, is assigned 16 addresses. It is a little trickier to keep track of these than it was with the $CN00 ROM addresses. Slot 0 is assigned addresses $C080 through $C08F (49,280 through 49,295), slot 1 gets $C090 through $C09F, slot 2 is $C0A0 through $C0AF, etc. Collectively, they can be referred to as "$C0n0" addresses and they are all fairly easy to pick out from Figure 22.1a.

Much like the Apple address decoding system described above for the I/O Select line, there is a second decoding system for the Peripheral Card I/O Space. Whenever an address between, for instance, $C0E0 and $C0EF appears on the address bus, the Apple's motherboard decoding system sends a signal to slot 6 called "Device Select." This is a signal which appears on pin 41 of the connector slot (see Appendix B).

If you look at the address $C0E0 laid out in binary form you'll get an idea of what this can mean for a peripheral card:

C	0	E	0
1100	0000	1110	0000

As long as any address beginning with $C0Ex has been called, the Device Select line to card 6 will be turned on. Therefore, only the last four address lines have to be watched. Combinations of on or off signals in these four lines can be used to operate various AND gates and OR gates, etc., to switch various functions on or off. This is a very important and distinct concept. These last four address bits are being used to throw switches rather than to identify some location within a memory chip.

In the //c, there is a system which is similar to the II/II+ and //e Device Select, and it is based on a Programmed Array Logic (PAL) chip called the General Logic Unit (GLU). Whenever the Memory Management Unit (MMU) in the //c detects an address in the lower half of the $C0 page (equivalent to the Built-In I/O Space) it turns on its SELIO line. It also turns this line on if it detects an address in the upper half of the $C0 page, but only for slot 1, slot 2 and slot 6.

The GLU detects this SELIO signal and looks at address bus bits 0, 3, 4, 5, 6 and 7 to decide what is being addressed. It can respond by activating the IWM disk controller (see Chapter 23), by selecting the two serial ports, or by passing the information along to the IOU via its own "IOU SELIO" line (see Appendix A).

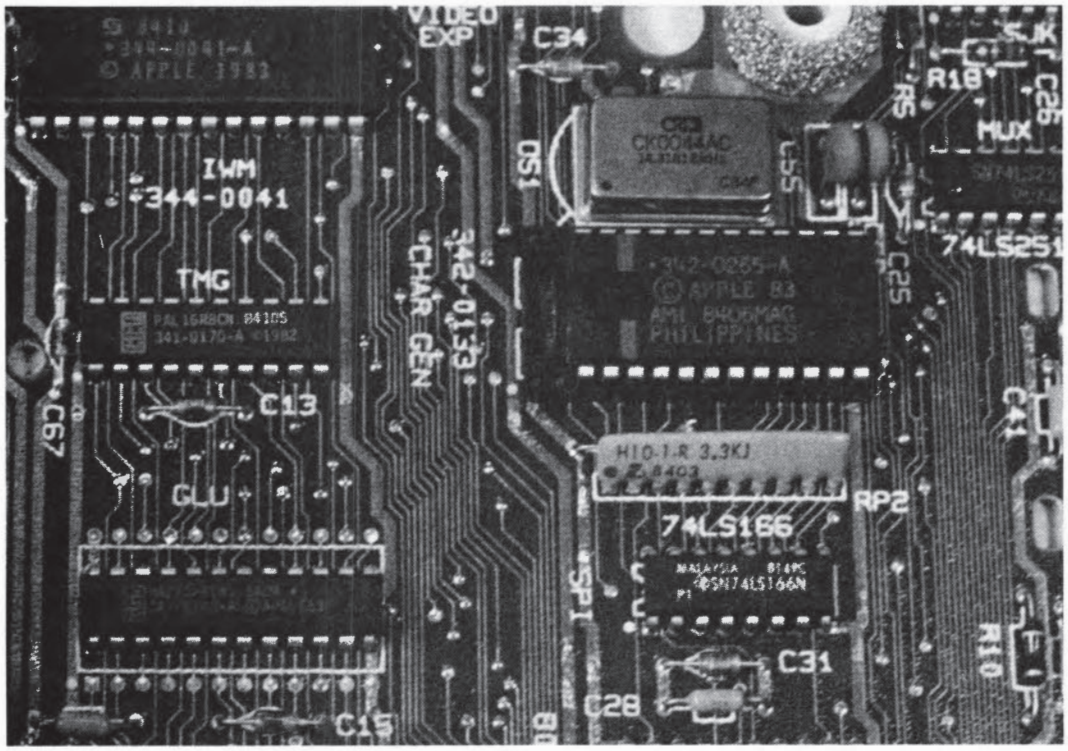
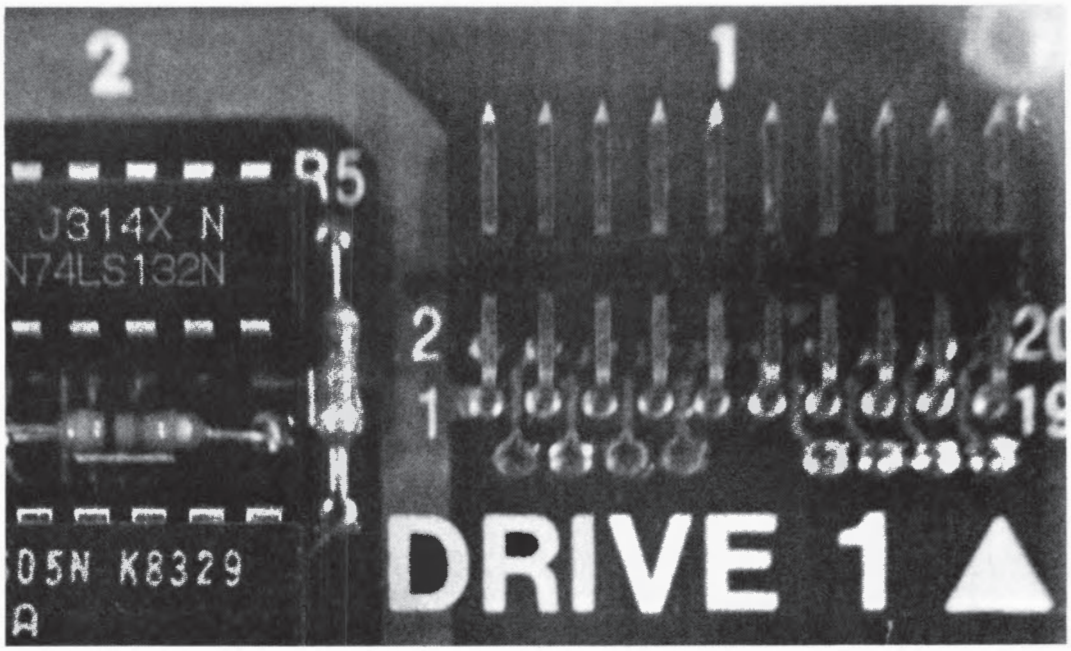


Fig. 22.9 The GLU (General Logic Unit) handles most of the I/O selection chores in the //c.


2
1
95
J314X N
N74LS132N
2
1
20
19
05N K8329
A
DRIVE 1

Chapter 23

Disk Drive Function

The Disk II Control System

To demonstrate how the Peripheral Card I/O space works, the Disk II system will be described in detail. It can serve as an example of a control/status/data port and the kind of functions which can be performed by sending signals to the lowest four address pins.

The other good reasons for pausing at this point to talk about disks and disk drives are that one, it is through location $C0EC/D (49,388/49,389; the effective data port on the disk II interface card) that most information passes in and out of most Apples in the world, and two, much of the rest of the discussion of the $C0 page has to do with using bank switched RAM in the place of disk drives. It is best to have a clear idea of the true meaning of disks before plunging on.

The Apple's Other Address Spaces

One recurring theme up to this point has been that the Apple has only one address space. This space is defined by the 65,535 possible numbers which can appear in the 6502's 16 bit address buffer (see Chapter 21). However, it would in fact be more accurate to say instead "The 6502 has only one address space." The Apple itself can have more than one.

Each of the Apple's other address spaces is defined by a simulated address register made out of two bytes of RAM. Into one of these simulated address registers you can, of course, put any number between 0 and 65,535. There can be several of these simulated address registers, each with its own 64K address space. Since these registers are not actually inside the 6502, they are managed by instructions to the Apple's operating system rather than by direct commands to the 6502.

Keeping Track of Megabytes

It would be nice, but no really big deal, if each location in one of these extra address spaces held a byte of information. You'd get supervision of an additional 64K bytes of data with each new simulated address register. In fact, the situation is much better.

ProDOS has some of these extra address spaces, but each location can contain 512 bytes instead of just one byte. In a full 65,535 location address space, with 512 bytes in each location, you

get a fairly good degree of supervision over 65,535 times 512 equals 33,553,920 bytes (32 megabytes). Even in 1984, very few microcomputer owners can afford that much RAM, so the standard way of taking advantage of these huge extra address spaces is to put disk drives into them.

Locations in the ProDOS/DOS Address Spaces

In ProDOS, each location is called a "block," and each address space is assigned to a "block storage device." If you want to use 64 megabytes of information, you're going to have to convince ProDOS that you have two disk drives and then it will happily set up a second address register to create a second extra address space.

There is a very similar system in DOS 3.3, but each location can only contain 256 bytes, so the maximum in one space is 16 megabytes. Another difference is that DOS 3.3 is very determined to think of anything in its address spaces as if they were disk drives. Much like the system in the 6502 address space, DOS 3.3 deals with its addresses as if they had two parts. The first byte describes what is called a "track," and the second byte describes what is called a "sector." The terms track and sector refer to details of the inner workings of disk drives and this makes it a little bit tricky to put RAM chips instead of a disk drive into a DOS address space.

Much worse than the expectation of a disk drive in DOS is the additional expectation that the disk drive will be a standard Disk II with just 35 tracks and 16 sectors on each track. This tends to limit you to 35 times 16 times 256 equals 143K bytes which just doesn't hold a candle to 32 megabytes.

That was fine in 1978, but that is also part of the reason that Apple is starting to abandon DOS 3.3 in 1984. Part of DOS 3.3 actually can handle big storage devices, but other parts of the DOS program aren't so willing. A variety of companies manufacture large capacity storage devices for the Apple, but each of them has had to write their own version of DOS all of which has resulted in an incompatibility nightmare. ProDOS alleviates all this.

File Buffers for Moving Data from One Space to Another

In any case, the way the whole thing works with ProDOS (or DOS or just about any operating system) is that some program such as a word processor or data base program asks ProDOS to see if it can find a chunk of information called a file. The program may have in mind a small file containing just two or three bytes, or it might be looking for a file containing a mailing list of all the Democrats in the state of Alaska (16 megabytes?). ProDOS is responsible for discovering whether such a file is within one of its address spaces and for preparing to move individual blocks from that address space into the 6502's address space.

Once such a 512 byte block has arrived in the 6502's address space, the word processor is able to move it around and feed it to other parts of the program. Note, however, that each individual location in the ProDOS address space will take up 512 locations in the 6502's address space the moment it arrives. In order to make sure there is room, ProDOS opens up what is called a "file buffer" which includes a string of 512 locations in the 6502's address space. ProDOS is always able to move the entire contents of one of its locations, en masse, into one file buffer.

This ability to move entire 512 byte blocks is very important because it means that ProDOS never has to worry about what is inside the block. It can treat it as one single entity. Just deliver it to the file buffer and then let the word processor worry about what's inside it. It is because of this breezy ability to abbreviate that ProDOS is able to keep track of over 33 million bytes in its address space while the 6502 can only deal directly with a little more than 65,000.

An Overview of the Disk Access Process

Given that ProDOS knows which block it wants to read, the process of actually getting the data from the block into the file buffer can be fairly involved. If the block storage device is actually some sort of RAM card somewhere in the Apple, then ProDOS can probably move the information by making the 6502 throw some bank switches and then telling the 6502 which area of switched RAM contain the blocks of data. There are a wide variety of these "RAM disk" schemes, all of which are reviewed a little later.

If, however, the block storage device is actually a disk drive, then ProDOS will have to use a special device called a "disk drive controller" to get at the data in the drive. ProDOS tells the controller the address of the block it's interested in, and the controller is supposed to be able to come up with the data.

The controller's task can be broken down into four steps. First, it has to prepare the drive mechanically, which can include both of turning on the drive motor to make the disk spin, and using a second motor to physically move a "read/write head" to a precise position over part of the disk. Second, it must sort through a long stream of incoming signals to determine when that part of the disk which is actually holding the specified block of data has spun into position over the read/write head.

Once all this positioning is complete, the controller must decode the incoming "raw data signal" to extract the actual bytes of information. This raw signal includes both data and various timing marks, and it must be cleaned up before it is handed to the Apple's data bus. As a fourth and final step, most disk drive controller systems are expected to assist in the transfer of the data into the ProDOS file buffer.

The RWTS/Disk II Interface Card System

A full featured disk drive controller card can do all of these tasks and more with remarkable efficiency, versatility and at very high speed. Most of the work can be supervised by a dedicated disk controller chip such as the WD 1797 from Western Digital. However, a full featured controller like this (i.e., the Vista A800/801) can be rather expensive ($550).

The Disk II Interface Card from Apple (see Figure 23.1a) is a little bit less than a full controller system. To operate, it requires the full and undivided attention of the 6502 under supervision of a special machine language program called Read/Write Track Sector (RWTS). RWTS gets involved with operating the disk drive at a very tedious level and it relies on such remarkable arcania as the exact execution time in microseconds of some of the 6502 machine language instructions in order to time events taking place near the surface of the spinning disk.

Wozniak wrote the first version of RWTS (on the back of an envelope just hours before the Disk II would be unveiled at a trade show, according to legend) when mini-floppy disk drives

were the hottest new wonder from Southern California. RWTS has been modified a few times since then, but it is in fact just one of those machine language wonders whose performance just about no one can improve on. And for the most part no one would want to since these things are all done automatically now by drive controller chips. In fact, Apple has been so happy with the system that it has now been frozen in silicon in the form of the Integrated Woz Machine (IWM) chip which is used as the disk controller device in the MacIntosh and in the //c (see Figure 23.1b).

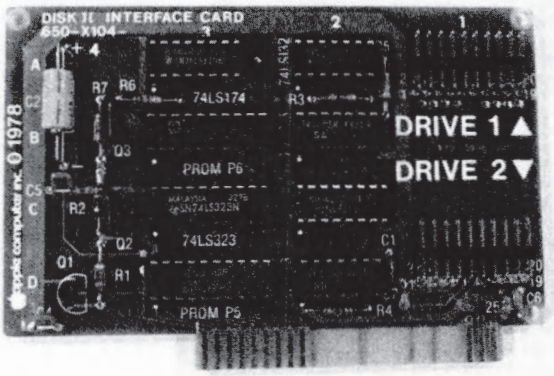


Fig. 23.1a Disk II interface card. The electronics are simple and most of the work is done from software.

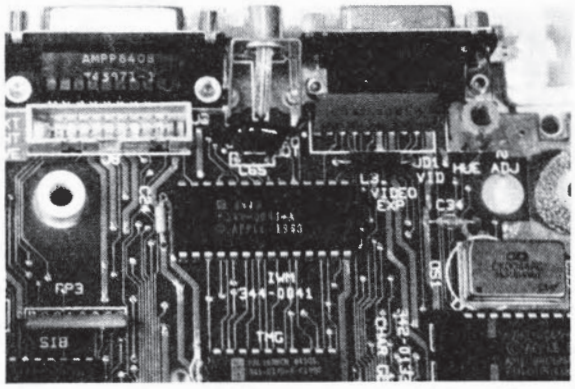


Fig. 23.1b The IWM (Integrated Woz Machine) chip replaces all of the circuitry on the Disk II interface card. The //c's internal drive plugs into a connector on the motherboard directly behind the external drive connector.

The advantage of the RWTS/Interface Card system has been that the card was very inexpensive. The disadvantage is that RWTS has gotten Apple DOS tied down in what are now fairly low capacity, low performance disk drives (i.e., the Disk II). ProDOS can use a version of RWTS when it works with the Disk II Interface Card and Disk II, but it has been designed to operate a full scale controller card if it finds one, including elaborate hard disk controllers. Similarly, a different software driver can use the same Disk II Interface/IWM hardware to operate much higher capacity drives than the standard Disk II or Disk //c.

Organization of Data into Tracks

The data stored on a disk is laid out in a perfectly round circle called a "track." There are always several of these tracks on a disk, all one inside the other as concentric rings (see Figure 23.2). The number of tracks squeezed onto one surface of a 5 1/4 inch disk ranges from 35 (the Disk II) to 306 (the Seagate 406 hard disk), but in all cases a track contains about 65,500 bits in a single "serial" stream. The Disk II interface card has to use about half of these bits as timing signals, so the Disk II gets about 4K bytes of data into a track. More advanced controllers can use nearly all of the bits for data so that a full 8K bytes can be stored in a track.

The timing problem in which the Disk II interface invests so many bits is a very tricky one. A disk drive reads data one bit at a time by the use of a "read/write" head. The read/write head stays in one place as the disk spins by, and the stream of passing bits is read off into the drive circuitry. The disk spins at a fairly steady speed so the bits of data come in at a steady rate.

Timing Tricks and Disk Data Encoding

In the Disk II, a drive motor spins the disk at 300 revolutions per minute, and in each track there are 65,000 data bit positions plus about 30,000 positions taken up in the necessities of formatting. This all means that a new bit passes by the read/write head about once every two millionths of a second (two microseconds). The controller can look at the bit stream every two microseconds and check whether a one or a zero has just come in as it works along reconstructing the stored bytes. It can reasonably expect to package up a complete byte in 16 microseconds and to send the byte off to the computer.

The problem is that it is not difficult to imagine little snags and warbles which could throw off the rotation speed by one or two millionths of a second. And, in fact, lots of things do throw off the rotation speed. Let's take, for example, the ASCII code for the letter O "0100 1111," and assume a snag occurs between the two zeros. If the controller just looked blindly at the data stream at an exact rate of once every two microseconds, it might think it was seeing three zeros (0100 0111) when in fact it was just a matter of the second zero arriving a little late.

And what if it happens to take 18 microseconds instead of 16 for a whole byte to pass by the read/write head? If the controller launches off the byte on its way to the computer after just 16 microseconds, it will be an incomplete byte. Worse still, it will then mistakenly consider that last late bit as if it were actually the first bit of the next byte in the stream thus producing a "framing error" which could garble all the subsequent data in that stream. But fear not, steps have been taken to prevent this particular set of disasters from occurring.

The way out of the bind is to provide the controller with an exact and instantaneous means of monitoring the actual speed of the disk drive before it reads each bit. With the Disk II interface card, this is accomplished by intermixing data bits with "clock bits," as shown in Figure 23.2 There will thus never be a chance for two zeros to come along directly in a row.

There are actually a few more constraints on which bit patterns are allowed. Those that are considered likely to confuse the controller are simply forbidden and a complex recording scheme is used to eliminate them. The consequence of this scheme is that out of every 16 bits in the stream, eight are clock bits, six bits are used for data, and two more are taken up by the coding system. This "six and two group coded recording" or "6/2 GCR" system is explained in some detail in *Beneath Apple DOS* by Don Worth and Pieter Lechner (Quality Software).

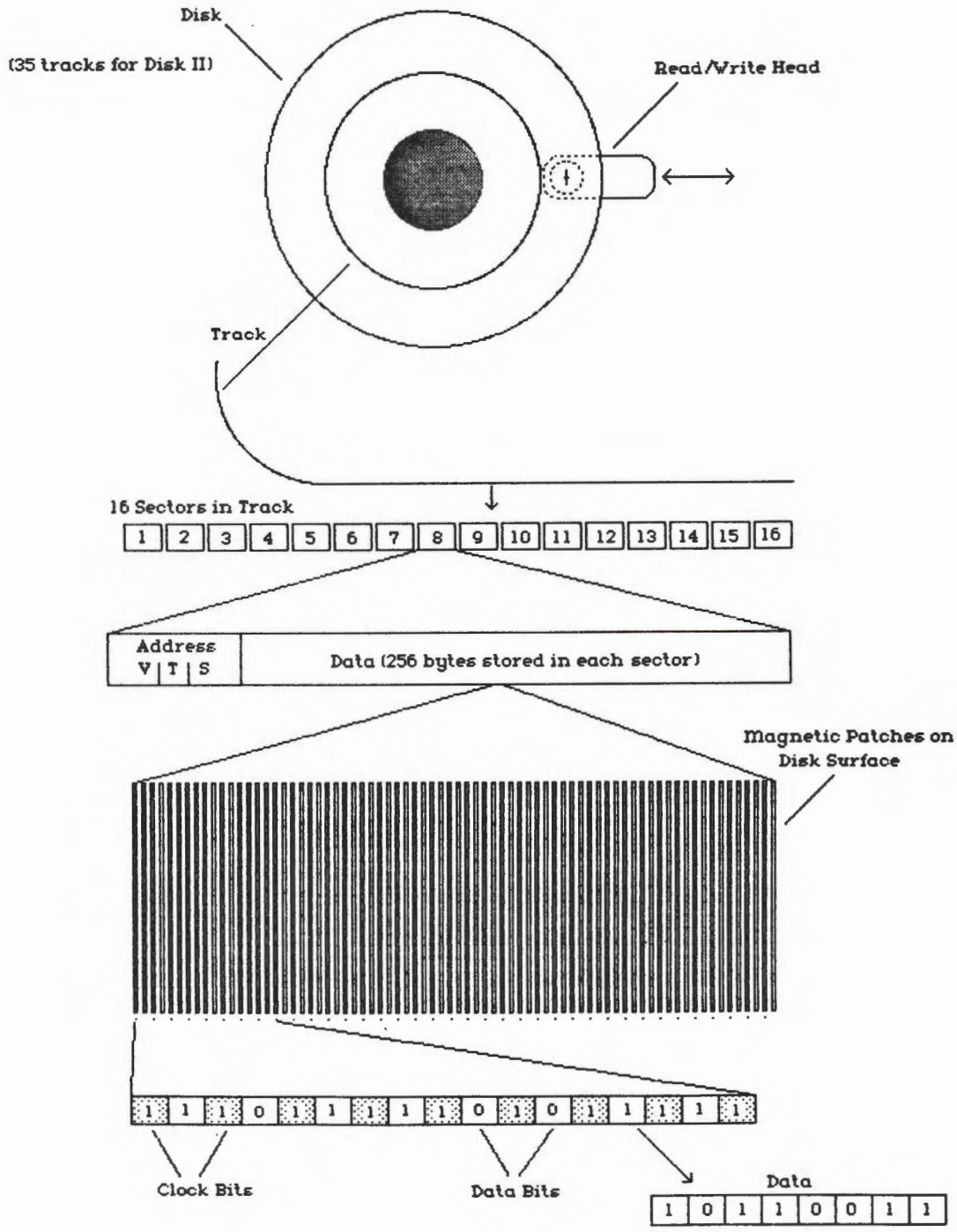


Fig. 23.2 Data is coded in 6/2 GCR format, interspersed with clock bits, and recorded as magnetic patches. Each of the 16 sectors in a track has an address field (V=volume, T=track number, and S=sector number) and a data field separated by a gap made up of synch bytes.

The Vista A800/801 floppy controller and most hard disk controllers use a more sophisticated clocking system called Modified Frequency Modulation (MFM). The electronics are a bit subtle and involve a kind of circuit called a "phase locked loop." The result of this system is that an MFM controller can detect extremely small variations in the time at which each bit arrives. A full explanation of MFM is beyond the scope of this book, but the result is that effectively all of the 65,000 bit positions can be used for data and none have to be reserved for clock bits. This is why each track in an MFM system holds 8K bytes of data rather than 4K bytes.

Although there are many manufacturers of Apple compatible floppy disk drives and disk drive controllers, nearly all of them have avoided using MFM systems in order to keep their controller cards inexpensive. Some of these systems pack more bytes onto a disk than a Disk II does because they can use more than 35 of these 4K tracks. The Vista A800/A801 MFM controller is used to operate "industry standard eight inch disk drives" (meaning that the same disks can be read by just about any microcomputer with an eight inch disk system) and high density 5 1/4 inch Amlyn drives (see Chapter 24).

Selecting a Track

Once ProDOS has decided which block of data it wants, it must figure out which track in which disk drive the block is stored in. It then orders the controller to prepare to access that track. The process of accessing a track is a mechanical thing, with spinning motors and sliding arms, but it must be done with great speed and enormous precision. In the worst case, you might have four Seagate 419 drives attached to a single controller card and therefore have to bring a read/write head over to one of 7,000 tracks. Even if you have just one Disk II drive with 35 tracks, the head positioning task is time consuming and demanding.

Drive Select

The first step in the sequence is for RWTS (or the A800) to activate one of the drives attached to the controller card. Hard disks can take between 15 seconds and a full minute to get up to speed. If you had to wait this long every time you wanted a little information, hard disks would be pretty impractical. Fortunately, in a hard disk, the read/write head floats above the surface of the disk on a cushion of air, so it is OK to turn on a hard disk and just leave it spinning for hours or days without fear of wearing down the disk surface. The process of "selecting" one of these drives is simply a matter of telling it that it is the drive which is supposed to respond to the next set of track commands.

In the Disk II, the disk is clamped directly into contact with the read/write head as long as the drive door is closed (see Chapter 1, Figure 1.6), so it is not wise to leave the disk spinning. Each time ProDOS selects one of the drives, its main motor is turned on and ProDOS waits for a little over half a second to allow the drive to get up to speed before it tries to read or write. The drive is then turned off again after each access to help extend the life of the diskettes.

The larger eight inch disk drives can take a full two seconds to get up to speed, so many manufacturers leave these drives spinning as long as the computer is on. To cut down on wear and tear they usually include an additional mechanical system which can lift the read/write head off the surface of the disk except for the few moments when a read or write is actually taking place.

You may recall that we had been talking about part of the Apple's $C0 page when we broke off to review disks. In particular, if you refer back to Chapter 22, you should remember that each of the Apple's slots is assigned 16 locations in the Peripheral Card I/0 Space. When a Disk II Interface Card is installed in slot 6, it uses these I/O locations in the fashion shown in Chapter 22, Figure 22.1a, which includes a blow up of the 16 bytes between $C0E0 (49,376) and $C0EF (49,391).

At this point you should pay particular attention to location $C0E9 (49,385) labeled "motor on." Those of you who like to actually try these things out should go to your Apples and get ready to make like RWTS by throwing a few switches on the Interface Card. Turn the Apple off and on and then hit reset (CTRL-reset) to make the drive stop spinning as usual. Now, respond to the square brackets prompt by typing: POKE 49385,0 and when you hit Return, the drive should turn on.

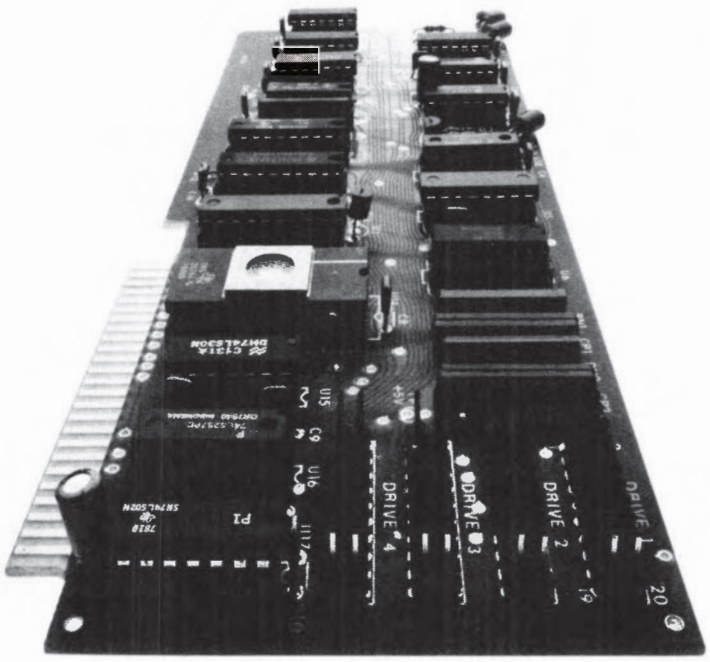


Fig. 23.3 This four drive controller from Rana is identical to a Disk II interface except that it extends the drive select function to handle more drives.

To stop it type: POKE 49384,0. The chart in Chapter 22, Figure 22.1a should give you enough information to also do the "drive select" function to turn on your other drive instead (see Figure 23.3). No other self-respecting disk drive controller card would let you do that sort of thing, but since the Disk II control system involves both RWTS and the card, it is a partially open system in which you can intervene (for educational purposes only, of course).

Stepping the Head

You can think of the tracks in a disk drive as a collection of 4K (or 8K) storage areas. The task during a track access is to move a read/write head into one of these areas. Although the tracks are actually concentric rings, the drive can bring the head into any one of the rings by moving it back and forth along a single line as shown in Figure 23.2. The act of moving between tracks is called "stepping," and it is therefore no surprise that the equipment which moves the head is called a "stepper motor system." A stepper motor is a fundamentally digital version of our old friend the electric motor. Instead of spinning continuously, its shaft turns in little tiny steps. A stepper motor has inputs for digital pulses and each time it gets a pulse, it turns its shaft one more step.

There are several ways of converting the stepping rotation of the stepper motor's drive shaft into linear motions of the read/write head, and Figures 23.4a and 23.4b show the system used in the Disk II. The mechanical guts of the drive are the standard Shugart SA400. The electronics have been modified by Apple in the Disk II, so you cannot just go out and buy an SA400 for your Disk II controller (Electrovalue sells a kit which lets you modify an SA400 for use as an Apple drive). The Apple //c has an ALPS drive which is very different mechanically from the Disk II, and has a much more precise head stepping mechanism (see Figure 23.4c).

To operate a stepper motor in the drive, an A800 controller system sends a direction signal which tells the device whether to move in or out from its current position, and stepping pulses which tell it how many tracks to step through. A full controller system such as the A800 remembers the current position of the read/write head. ProDOS tells the controller which track number it would like to use next. The controller calculates the necessary direction signal and the number of step pulses.

The Disk II Interface Stepper System

If you look back at Chapter 22, Figure 22.1, you'll see that the addresses between $C0E0 (49,376) and $C0E7 (49,383) are labeled as Phi 0 on/off, Phi 1 on/off, etc. These four pairs of locations are the stepper motor controls. The symbols stand for Phase 0, Phase 1, Phase 2 and Phase 3. Each of these motor phases has an on and an off position. When RWTS is given a track number it decides on a direction and then starts fiddling with these motor phase addresses.

To turn the stepper motor shaft, RWTS addresses each of these locations in order. To move the head in toward the center of the drive it calls $C0E0, 1, 2, 3, 4, 5, 6, 7, 0, 1, etc. To move the head back towards the outside, it addresses them in reverse order, $C0E7, 6, 5, 4, etc. To step from one track to the next, it has to turn two phases off and then on. There are therefore two steps per track. If you only do one step, then you are between tracks—a place called a "half-track." Some copy protection schemes take advantage of this "half tracking" capability to put crucial pieces of information in places where RWTS and most copy programs can't find them. The protector then provides a special version of RWTS which can get at the half tracks.

Limits on Tracks Per Inch

Although the stepper system in an SA400 is capable of moving to 70 different but narrow track locations, there are two factors which limit the SA400/Disk II to just 35 wider tracks of usable information: the accuracy of head positioning, and the sensitivity of the read/write head.

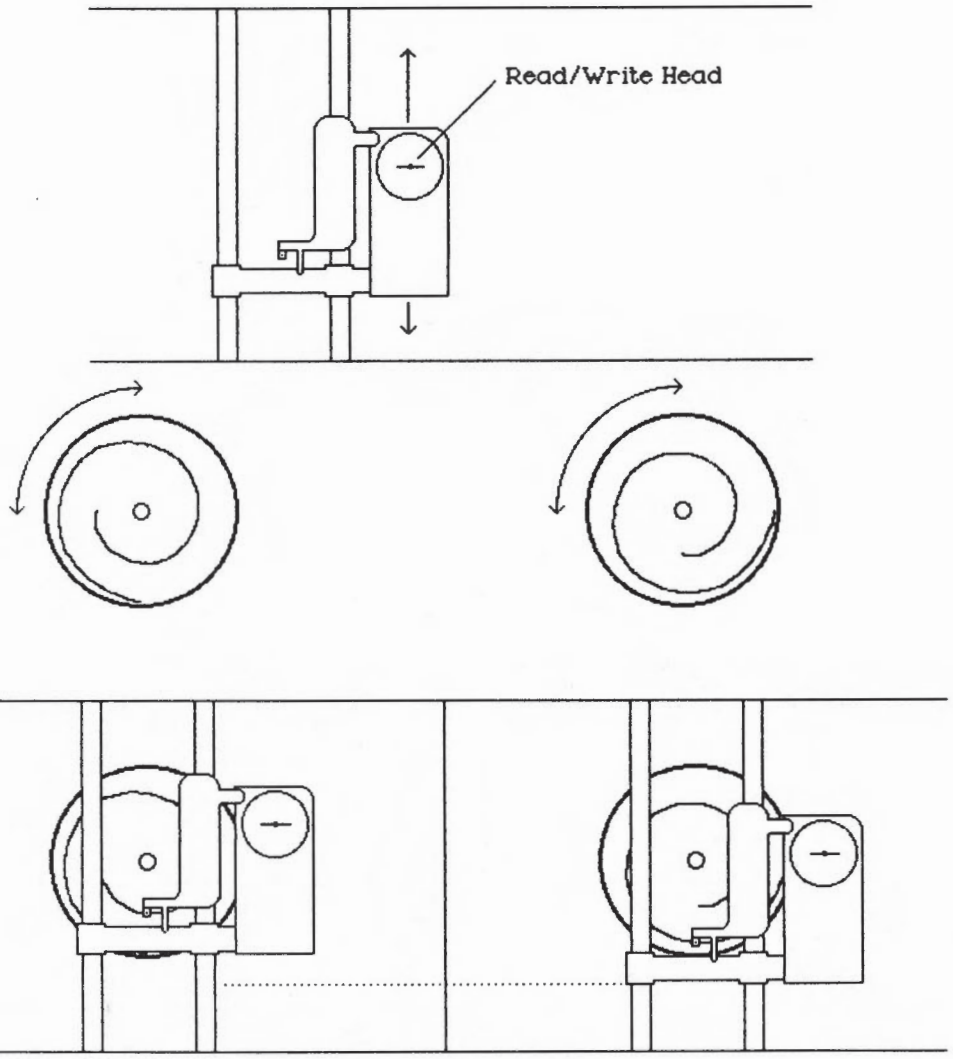


Fig. 23.4a Disk II head positioning mechanism. The stepper motor rotates a wheel with a special spiral groove into one of 35 different positions. The read/write head slides in and out from the center of the disk along guide bars. Its position is controlled by the spiral groove.

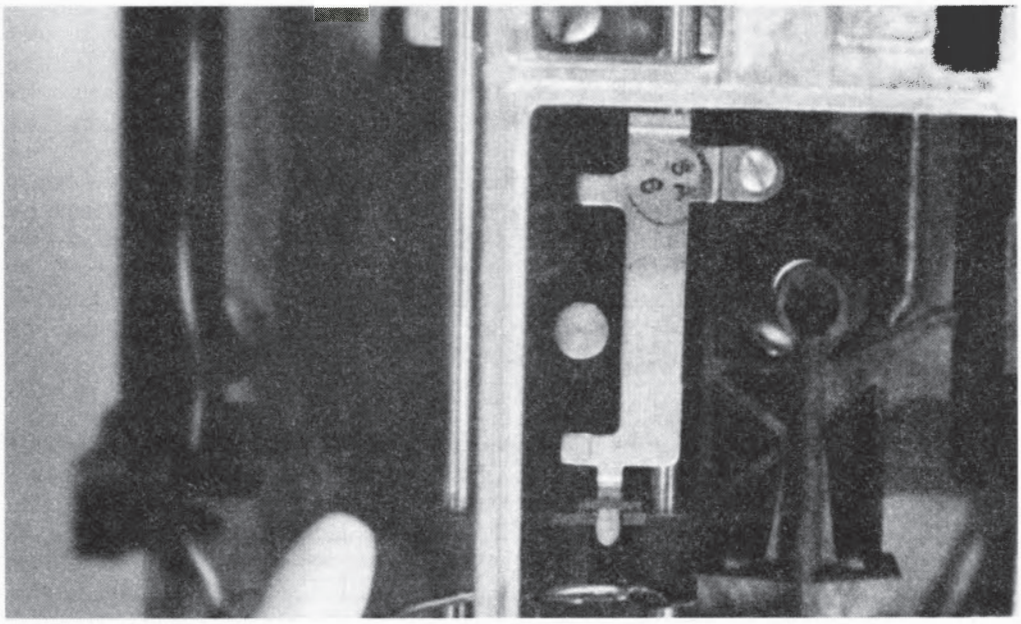


Fig. 23.4b Disk II head positioning mechanism.

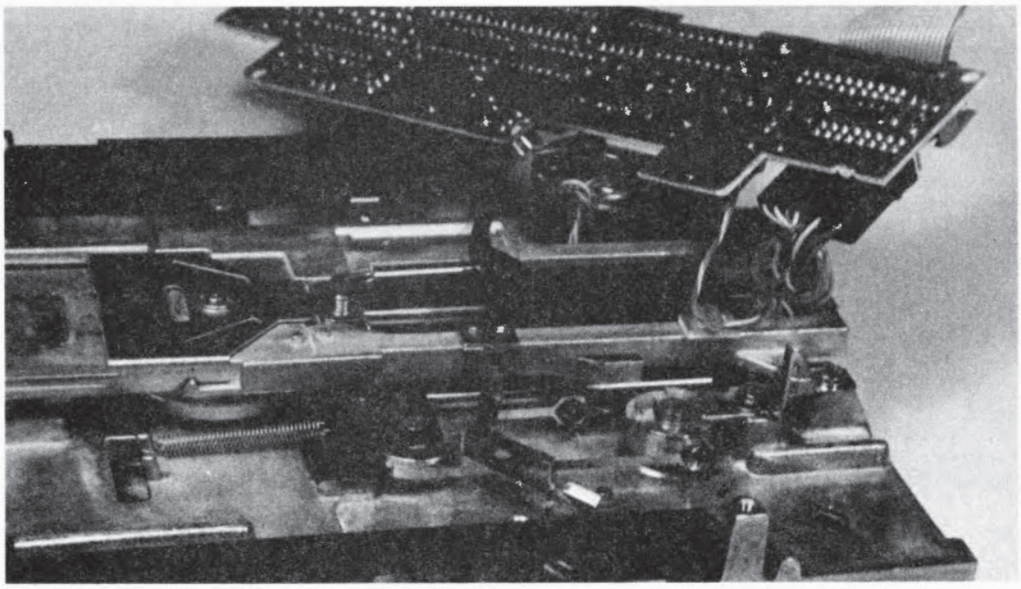


Fig. 23.4c Disk //c head positioning mechanism. This is called a band positioning system because of the metal band that runs around the two metal wheels. This is a much more precise system than the Disk II's spiral, but the differences conflict with some hardware based copy protection schemes used on disks for the Disk II, causing some disk incompatibility between the two different types of drives.

When the fully coded digital signals arrive at the read/write head for writing, the head converts the pulses into little panels of magnetic charge. Each pulse is stored as a pair of opposite magnetic signals. At each panel, the first half of the little bar magnets are made to point south, followed by a second bunch pointing north. When the read/write head passes over the pool later for a read, it senses this "flux change." The inside and outside edges of each track are simultaneously trimmed by a pair of erase heads so that a neat, clean track is laid out.

To help assure the reliability of reading and writing, the designers of this system wanted to use the smallest possible panel in order to get the most bits on the disk. However, the smaller the panel, the weaker the signal and the greater the chance for error. Each panel is therefore laid out as a rectangle which is more than 100 times wider than it is long (see Figure 23.2). The SA400 can pack in these flux change panels at a density of over 5,000 per inch (5,000 fci) when they are stacked side by side, but when they are stacked along their long sides, the density is 100 times lower, 48 tracks per inch (48 tpi).

More Tracks in New Drives

There has been only a little progress during the past six or seven years in increasing the Flux Change per Inch (fci). Newer hard disks and very high density floppy disks get by with about 9,500 fci. However, there have been dramatic improvements in the number of tracks per inch. The new 3 1/2 inch microfloppies, used in the MacIntosh, are recorded at a density of 135 tracks per inch, and the highest tpis among 5 1/4 inch floppies are provided by drives from Drivetec and Amlyn, which support 170 tpi. The Amlyn mechanism became available to Apple owners in 1983 as the Vista V1200 and it can store 1.2 megabytes on one side of a 5 1/4 inch floppy, The Drivetec system is sold by Rana as a two sided drive (see Chapter 24) that gets 2.5 megabytes onto a disk. The new 3 1/2 inch Sony floppies provide 400K bytes on one side of the disk in a very convenient package.

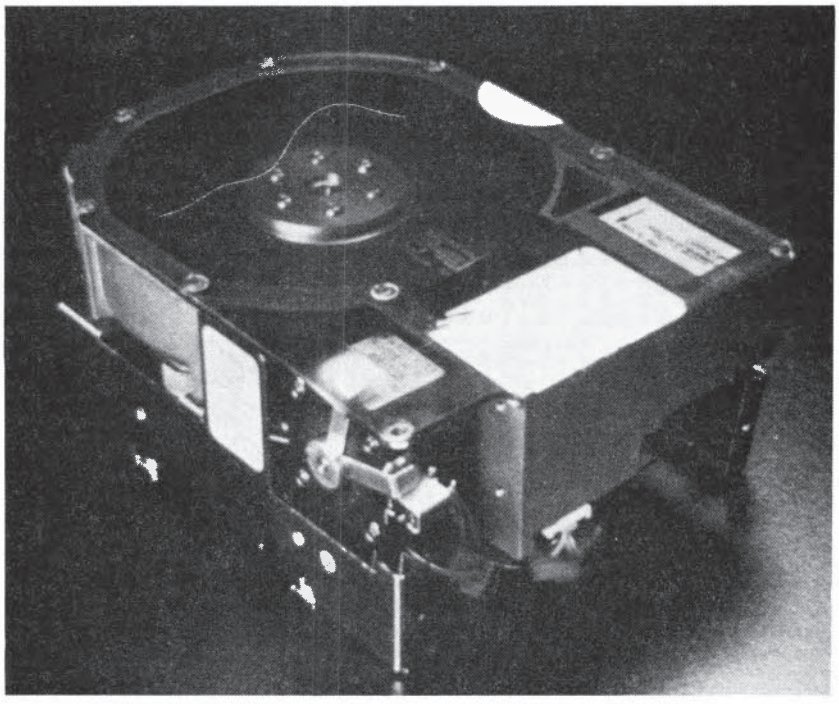


Fig. 23.5 Inside a Profile hard disk.

The fact that floppy disks are floppy puts some limits on the tracks per inch. They are subject to various deformations and shape changes which make high tpi very challenging technologically. It has been possible to build 170 tpi hard disks since 1978 (Shugart SA1000 eight inch hard disk), and the very popular Seagate 506 (see Figure 23.5), a 5 1/4 inch hard disk, has 260 tpi. The newest hard disk drives (Shugart SA706 series and Seagate ST 406 series) achieve densities of 345 tpi. (In case you're getting confused between Shugart and Seagate, you should know that the president of Seagate is Alan Shugart.)

Traveling Time for Stepping Heads

Unfortunately, stepper motors do not move nearly so fast as microprocessors. In fact it takes about 20 milliseconds for each track to track step in the Disk II, and once the read/write head arrives at the proper position, it takes another 15 milliseconds for it to stop shaking and settle down to work. That comes to 35 milliseconds for one track. If the Disk II head has to travel from track 0 to track 35, it will take 715 milliseconds, or nearly a second. Throw five or six of these into a file read, and you've got a three or four second wait for just a few blocks of data.

DOS and ProDOS try to get around this by saving related blocks of information close together on the disk, preferably all in a single track or at worst in two directly neighboring tracks. The new Seagate 419 hard disk packs in 1,836 tracks, so what was a problem with 35 tracks would be really big trouble in this hard disk (maximum of 35 seconds to change tracks?). Fortunately, the Seagate 406 handles things a good bit more elegantly.

One improvement is that the "track to track access time" is only three milliseconds instead of 20 milliseconds. This "three ms" stepping rate is actually a holdover from years past, since eight inch floppies and hard disks have been using 3 ms stepping rates for years. In fact, the stepping rate is the most serious drag on the performance of many hard disk systems. The settling time on arrival is also an unimproved 15 ms.

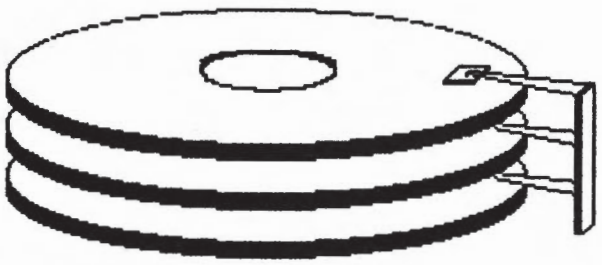


Fig. 23.6 Hard disk platters.

One of the ways that Seagate squeezes higher performance out of the system is to organize its 1,836 tracks into just 306 concentric "cylinders." Each cylinder is sliced by three disk platters, and there is a recording surface on the top and bottom of each surface (see Figure 23.6). The read/write head system actually has six heads, each assigned to its own disk surface and all of which move together.

The other major improvement is to use a "smart" disk drive which intercepts the stepping signals coming from the controller and calculates a smooth acceleration, glide and deceleration for long jumps. As a result of these and other tricks, the actual maximum, worst case access time, including settling, is 205 ms. This maximum access time of 205 ms for any one of 1,836 tracks compares very favorably with the Disk II which can take 715 milliseconds for 35 tracks. In fact, this is an improvement of about 200 fold and this factor is the major speed advantage of hard disks, access to very large amounts of information very rapidly.

The typical "figure of merit" that is used to describe all this is not the maximum, but the "average access time" which estimates how long a typical access will take. The Disk II average access time is given by Shugart as 275 ms, while the average access time for the Seagate 406 is just 85 ms. This means that for steady use of a limited amount of information the actual access by a hard disk is only about two or three times as fast. The true advantage of the hard disk is that this average access time stays the same despite the addition of 1,800 more tracks.

Picking Out a Sector

All of the maneuvering with drive selection and track access described up to this point has finally prepared the system to start reading in bits from one of the tracks. As you will recall, a track contains about 4K bytes of data (or 7 to 8K if you've got an MFM controller card), and if the read/write head just stays where it is, it will collect a fairly continuous stream of bits, with the whole thing repeating every time the disk finishes sweeping by in a complete circuit.

The objective, however, was to fill up just one file buffer in the 6502's address space. In DOS 3.3, the file buffer could hold just 256 bytes and so the 4K in an Apple Disk II track is laid out as 16 "sectors," each containing 256 bytes. These sectors would appear as short segments along the track (see Figure 23.2). Although ProDOS uses 512 byte file buffers, it organizes its tracks into 256 byte sectors, which helps make it much easier to convert DOS 3.3 files into ProDOS files. When ProDOS needs to read a block, it reads in two of these 256 byte sectors.

Formatted Disks Have Addresses for Their Sectors

When the read/write head arrives at a spinning track, there is no way of predicting which sector will be passing by at that moment. The read/write head must therefore sit and wait, scanning the data stream until the sector it wants spins by.

When the disk is first formatted, ProDOS records a unique track and sector number at the beginning of each sector (see Figure 23.2). Later, when ProDOS wants to do a read or a write, it tells RWTS (or the complete controller) the address of the sector it wants to use. The controller system scans the stream until it sees the appropriate sector address and then begins its read or write. If you want full details on the contents of the address label, you should consult *Beneath Apple DOS* by Don Worth and Peter Lechner.

The worst possible case in terms of time would be the situation in which the desired sector had just gone by as the head arrived at the track. In this case, the head would have to wait patiently, reading through 15 sector addresses before the correct one finally came around again. This waiting time is called "latency," and it is determined directly by the speed at which the disk is spinning.

The standard speed for 5 1/4 inch floppy disk drives is 300 rounds per minute (rpm) and this results in an "average latency" of 100 milliseconds. The new 3 1/2 inch drives are sold in several versions; one runs at 300 rpm, while a second version, used in the Hewlett Packard computers, is spun at 600 rpm. In the MacIntosh, the 3 1/2 inch drive has a variable speed control; 600 rpm at the inside, 390 rpm for the outermost track. This helps maintain the same information density (fci) on inner and outer tracks. The speed for eight inch floppies is always 360 rpm (average latency of 83 milliseconds), but hard disk drives are operated at the much greater speed of 3,600 rpm. This is why it takes 10 to 15 seconds to get one of these drives up to speed, but it also means that the average latency is cut to just 8.3 milliseconds.

The total time to get to a sector therefore includes motor start up (if applicable), stepping time, settling time, and latency. If we assume that the Disk II was already on and spinning at the start of the access, then average access time (275 ms) plus average latency (100 ms) comes to 375 ms for each sector that is read. A hard disk drive's total comes to just 85 ms plus 8.3 ms equals 93.3 ms. You have to turn on the Apple drive at the beginning of a read, so you must add another 500 milliseconds and then the comparison is 875 ms versus 93 ms to get at the first sector in a read.

Sector Skewing for DOS, ProDOS, and Pascal

Physical Sector	Logical Sector	
	DOS 3.3	ProDOS/Pascal
0	0	0
1	7	8
2	E	1
3	6	9
4	D	2
5	5	A
6	C	3
7	4	B
8	B	4
9	3	C
A	A	5
B	2	D
C	9	6
D	1	E
E	8	7
F	F	F

Table 23.1 Sector Skewing. The sector addresses in a track are scrambled to improve reading efficiency. When ProDOS needs block 0, it reads logical sector 0, misses a sector, and then reads logical sector 1. This minimizes the effects of latency in a block read.

Skewing of Sectors for Improved Performance

ProDOS achieves some improvement in performance through skillful use of a technique called skewing or interleaving. When a drive reads in a 256 byte sector, the Apple requires a few milliseconds to consolidate the data in the Apple's memory. By the time it goes back to the drive, it will be too late to spot the beginning of the next sector. If the next bunch of data it needs is in that next sector, then it would have to wait 200 milliseconds for a full rotation.

To avoid this problem, data is recorded in order but in every other sector. As a result, although ProDOS will not yet be ready when the next physical sector spins by, it will be ready when the one after that appears, and it is this one which actually contains the second half of the block. The ProDOS interleave scheme is laid out in Table 23.1. It is the same as the scheme used for Apple Pascal, but it is different from the DOS skewing arrangement.

Getting the Data Into the File Buffer

Once the controller system has accessed the correct track and located the desired sector, the bytes can start streaming their way towards the file buffer. In the Disk II and Disk //c, the first step is to pass through a "floppy disk read amp" chip called the Motorola 3470 on the analog board inside the drive cabinet. This chip is used in both the Disk II/SA400 and in the Disk IIc/ALPS drives. The chip converts the complex frequency and amplitude information of the read head data stream into a series of digital clock and data bits.

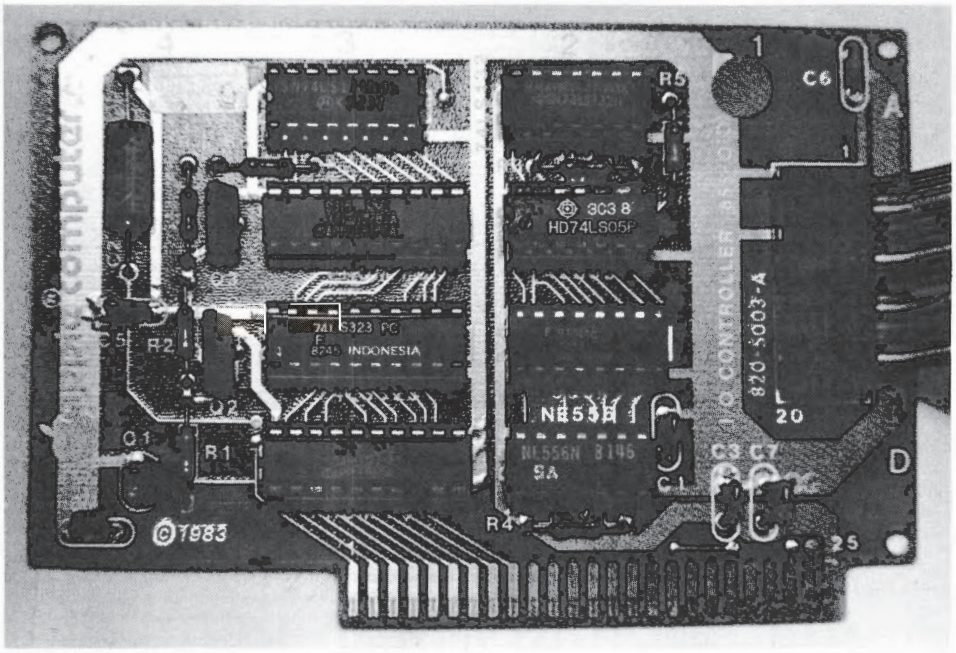


Fig. 23.7 Duodisk controller. This is identical to the old Disk II interface card, but it is impossible to plug the drives in incorrectly.

This digital stream is sent through the drive cable to the Disk II Interface Card (see Figure 23.7) in the Apple. There, a very strangely configured ROM chip called the P6A "state machine" uses a variety of tricks to pick the data bits out from among the clock bits (see *Understanding the Apple* by Jim Sather). Finally, the serial stream of data is loaded into a shift register one bit at a time to assemble a full parallel byte. The IWM in the //c works in a functionally identical way, but the internal details have been changed slightly. These slight changes cause trouble only for some of the most elaborate "copy protection" schemes.

Looking back at Chapter 22, Figure 22.1a, which displays the addresses for the controller card, you can see that the highest two bytes are labeled "set for output" and "set for input." To read a byte, RWTS sends a pulse to $C0EE (49,390) simply by addressing it, and this causes the data port to be "configured" for a read. Next, RWTS does an address to $C0EC (49,388) and this causes the shift register to dump its byte of data out onto the Apple data bus where the 6502 can grab it for further processing by RWTS.

When RWTS is collecting in data, it must go through this sequence 256 times at exactly the same rate at which the bytes are arriving at the register. Similarly, during a write operation, RWTS must set $C0EF and then keep hitting $C0ED as the register is fed its 256 bytes at a very exact rate tied to the rotation speed of the disk drive.

The Data Bottleneck for High Speed Drives

Disk drive controllers which do not use RWTS need to handle the data transfer a little differently. Hard disk controllers usually have a 256 byte RAM "sector data buffer" into which decoded data is dumped. DOS must then be modified to manage the exchange between the file buffer and the controller's sector data buffer. ProDOS can handle these sorts of special requirements much more gracefully since it only expects to use RWTS for a Disk II type drive.

In either case, this process of transferring the data from the controller's sector buffer to the Apple's file buffer can completely destroy any speed advantage the hard disk might otherwise provide. When the ad tells you that a hard disk can transfer data at five million bits per second, this only means that it can get the data into the controller's sector buffer at that speed. Moving the 256 bytes of data from the sector buffer can take five milliseconds, so that your actual rate of delivering data to the Apple is about 500K bits per second, 10 times slower than advertised.

But that's not the worst of it. Apple DOS can take its own good time about emptying the file buffer to prepare for the next read. The ultimate effect may be that data transfer from your very high powered hard disk takes place at the galling speed of about 5,000 bits per second. That's right, a thousand times slower than advertised!

ProDOS and CP/M are streamlined and do a little bit better, but the operating system software in the Apple itself can tower above all other factors in determining disk reading speed in the Apple.

DMA and High Speed with the Vista A800

There is a way out of this speed bind. Folks who use the Vista A800 controller card (see Figure 23.8) with an eight inch floppy system or high density Amlyn floppy can read data into the Apple 20 times faster than our hapless hard disk because of two tricks. The first is to skip the slow transfer of information into Apple memory by using a process called Direct Memory

Access (DMA; see Chapter 27). Simply put, the A800 uses some special circuitry on the Apple motherboard to disconnect the 6502 and seize control of the data and address buses. It determines beforehand where the data should go, and then pumps it in to its destination byte by byte as soon as it arrives from the disk drive. In this fashion, data is delivered directly to the file buffer as fast as it is read off the disk.

The second trick is a bit of software called Quickcharge, which bypasses the slow process by which DOS moves the data out of the file buffer to its intended location in the 6502 address space.

The Vista A800/801 hardware and software system points up a very frustrating aspect of the market for high density and high speed disk storage products for the Apple. A company may be the country's leading manufacturer of disk drives or disk drive controllers, but if they don't know the Apple inside and out, their product will have miserable performance. Thus there's a lot of drive hardware out there that works OK, but that comes nowhere near full potential.

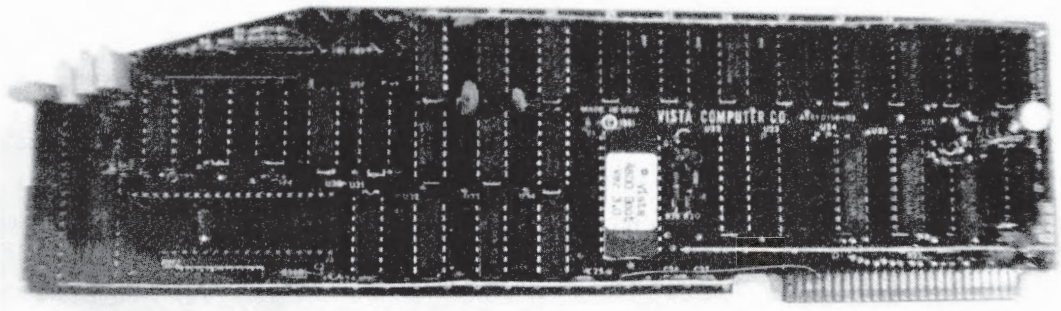


Fig. 23.8 Vista A800 MFM controller for 8 inch disk drives.

Write Protect

The only address in the Interface Card's address area which hasn't been covered yet is $C0ED. When the input signal at $C0EE has first been toggled for a read, then $C0ED can signal whether or not there is a write protect tab on the diskette. In fact, when there is such a tab on the disk, a switch is thrown (see Figure 23.9) and the electronics of the read/write head is partly disabled inside the disk drive, so there is no way to override the write protect feature from software. The reason why the write protect status is also reported to the Apple is that this is the only way the Apple can find out that its attempted write has failed without actually trying to reread the data.

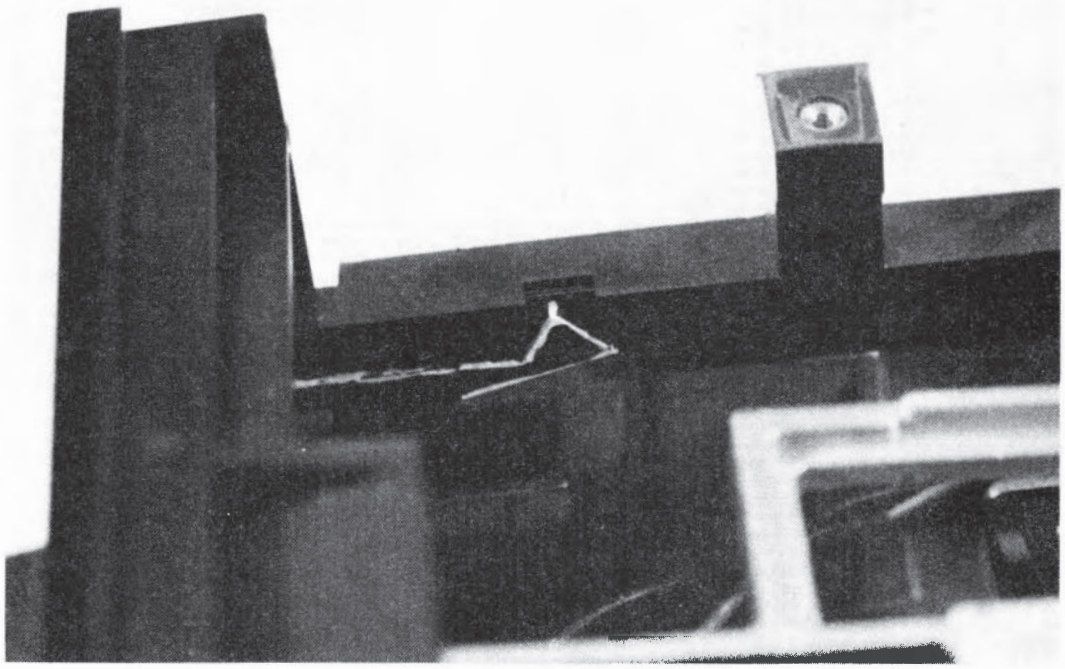
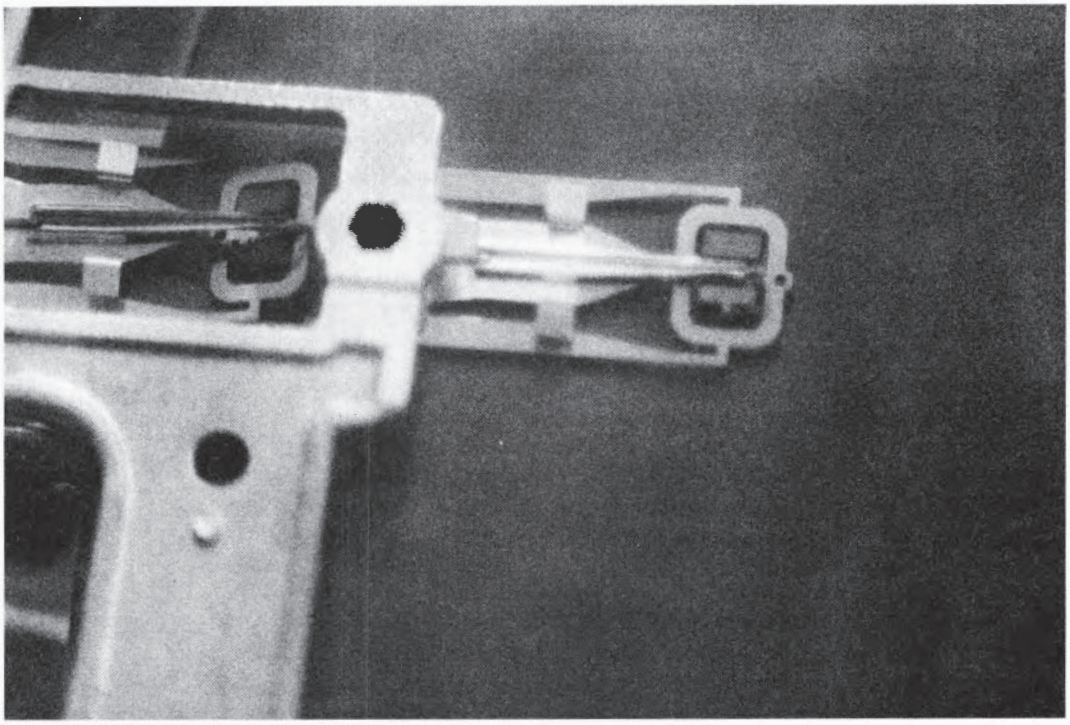


Fig. 23.9 Write protect switch. The lever is pushed down while the disk is sliding in, but then pops back up into the notch on the side of the diskette. A write protect tab prevents the lever from popping up.



Chapter 24

Getting More Disk Capacity

Buying a Disk Drive System

One of the best investments you can possibly make to enhance your Apple is to buy a second disk drive. The two principal benefits are that one, you will find it much easier to make backup copies of your work (the disasters you can avoid through careful backup can double or triple the worth of your system), and two, there are many powerful programs available for the Apple which you cannot use unless you have two drives.

The two broadest categories of additional drives are an Apple Disk II (see Figure 24.1) or Disk II compatible drive and a drive system that offers some sort of enhanced features but does not offer full Disk II compatibility.

If the principal use of your Apple is with CP/M (preferably PCPI CP/M which is easiest to adapt for non-standard drives), then just about any kind of disk drive or related mass storage device will be compatible both with your software and with any second or third unusual storage system you choose to buy. You are also in a reasonably good position if you use ProDOS or Microsoft CP/M software.

However, if you are not using a Z-80 card of some sort, watch out. DOS 3.3 is allergic to non-Apple drives. Those of you who only use your Apples for writing your own BASIC programs may find DOS 3.3 much more flexible, but if you use commercial software tread with great caution.

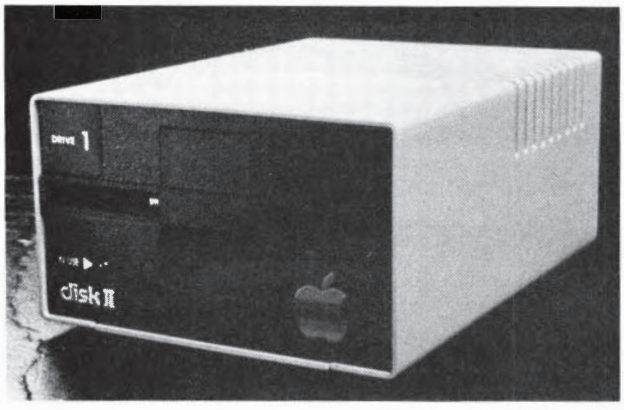


Fig. 24.1 Disk II.

The Disk II/RWTS System Demands Apple Compatible Drives

If you ran through the little exercise in Chapter 23, you can appreciate the fact that the Disk II drive control system is open to fiddling. Such fiddling is done quite often by commercial software companies and is better known as "copy protection."

Most commercial Apple software is copy protected, and this usually means that it takes advantage of subtle details of the RWTS program or of the Disk II drive mechanics to make it difficult to read the software in any way other than the way it was intended, i.e., for use and not for copying. An unfortunate side effect of these protection schemes is that you cannot use the software for its intended purpose if you try to use a disk drive system which does not use the complete Apple system (RWTS/Apple Disk Interface Card/Disk II). This is why some "protected" games and business packages won't boot on the //c drives unless you get a "//c version."

Because the Disk II is now a relatively low capacity, low performance system, Apple released ProDOS, which can accommodate many of the newer high performance drive systems and can improve the performance of the Disk II. However, there is no reason to assume that a particular piece of software will work with a particular disk system unless the respective manufacturers have tried it out and are willing to make claims for compatibility.

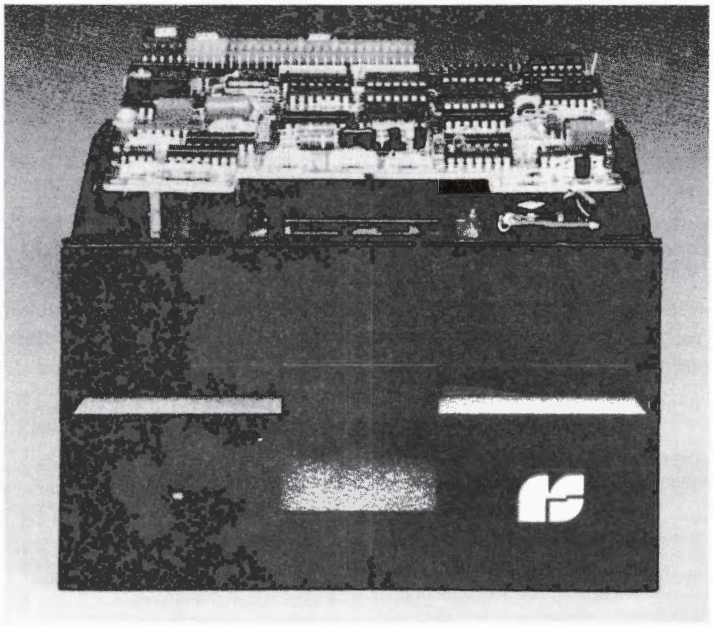


Fig. 24.2 SA400 drive from Rana.

Disk II Compatible Drives

The Apple Disk II is a modified version of the Shugart SA 400 disk drive. The exact performance of a Disk II is due to both of the unique aspects of the SA 400 (see Figure 24.2) mechanism and the special features of the Apple "analog board" inside the drive cabinet. Although there are many "Apple compatible" drives on the market, very few of them work just like a Disk II.

The term "Disk II compatible" seems to have come to mean that the drive in question; one, can be plugged directly into the Disk II interface card, and two, can be operated by DOS without any special "preboots" or changes in the software. Drives which meet these conditions sell for as little as $200 and are manufactured by a variety of companies. All of these drives will work reasonably well, but most of them will not "boot" copy protected software. This is because few companies take the time to adjust various electronic components on the analog board to properly duplicate all the quirks of the Disk II.

The only compatible drive which has gained a really strong reputation among dealers for full compatibility is the MicroSci A2. However, this drive lists for $479, about the same as a Disk II. Apple has not always produced enough Disk IIs to supply everyone who wants two of them, and MicroSci has been taking up the slack. Similar drives from Fourth, Quentin, Mitac, and Vista (the Solo) cost about half as much and work fairly well for most purposes, but don't expect to be able to boot Frogger (this, by the way, is the litmus test for full compatibility and "boot drive" status).

This situation is less critical for add-on drives for the //c. You will almost always use the built-in drive as your boot drive. If you do have a system that lets you boot from an external drive, then that system must have its own, specially modified, version of ProDOS.

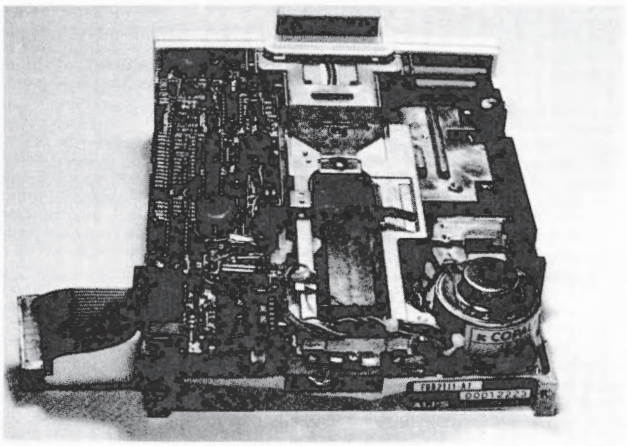


Fig. 24.3a ALPS drive mechanism used in //c internal drive.

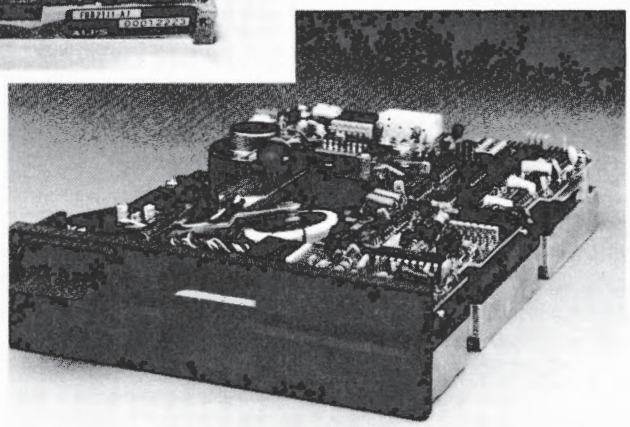


Fig. 24.3b Half-height drive mechanism manufactured by Qume. Both of these drives are half-height, but the internal design is completely different. If you want full //c compatibility, you may need to get an ALPS mechanism.

Fig. 24.4a Duodisk half-height drives.

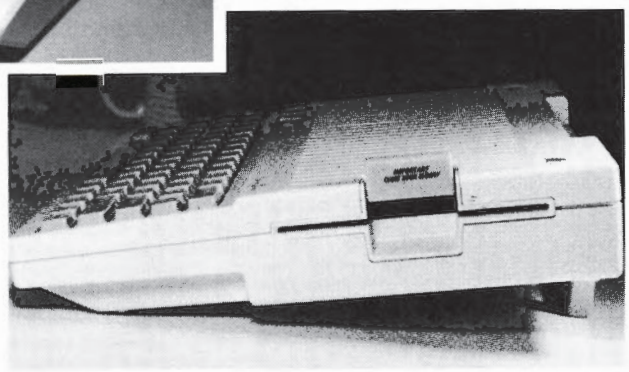


Fig. 24.4b The //c built in drive.

Disk II Compatible Drives with Added Features

Several companies offer drives which can be operated in two modes, either as a strict Disk II-like drive or as an enhanced drive. However, to use any enhanced feature, you've got to modify DOS and this means you can't use copy protected software. Once again, CP/M users are in good shape and ProDOS users have a good chance. Unfortunately, the "Disk II mode" on these drives is not always a perfect replication, although you're probably OK as long as you have a real Disk II for a "boot drive."

The added features available with various drives include increased storage per disk, faster access time, and smaller physical size for the drive. The half-height or "thinline" drives are manufactured by several companies (see Figure 24.3), the most prominent of which are Teac and ALPS. The Disk //c and the DuoDisk (see Figure 24.4) use slightly different mechanisms, and other drives are bundled and sold by Vista (Duet and Quartet), Lazer Microsystems, Comrex and Mitac.

The problem with these half height drives is a bit ironic. They are very well designed and they spin the disk at a very precisely controlled speed. The Disk II, however, has some characteristic irregularities in its speed control, and some copy protection schemes take advantage of these irregularities. This renders the Teac drives "too good" to boot some protected software.

IBM PC Drives, Quadlink and Rana

The Quadlink ($680) is a complete Apple computer which is built on a card that can be installed inside an IBM PC (see Chapter 29). The installation procedure involves intercepting the drive control cables inside the PC so that the Quadlink can sieze control of the IBM drives. This is a very sophisticated hardware and software system whose principal design objective was to be able to boot copy protected Apple DOS software. For the most part, Quadram Corporation has succeeded admirably.

This system does not permit the PC itself to read Apple diskettes. When the Quadlink board is active, the IBM PC is usually disabled. Only the second version of Quadlink makes it possible to pass information from the Quadlink environment to the regular IBM environment.

There are two ways to actually exchange disks between an Apple and an IBM PC. One method is a bit of overkill in that it involves purchasing eight-inch disk drives for both systems. You have to use CP/M 86 on the PC and purchase a Vista Diskmaster and an eight-inch drive system. This permits direct exchange with a similarly equipped Apple or any other eight-inch CP/M 80 system for that matter. The second and much more reasonable approach is to get the Rana 8086/2 system (see Chapter 30). This is no less expensive, but it lets you run regular PC disks on an Apple peripheral as well as adding a complete PC compatible computer to the Apple. In this case, though, you can't count on any existing utilities to copy your data stored in DOS into the IBM format.

Although there is a great deal of game and educational software available only on Apple disks, the principal use for the Quadlink will probably be for folks who have an Apple at home and a PC at work or vice versa; you can take your Apple VisiCalc work from one machine to another. The powerful statistical packages which run in Apple Pascal are steadily migrating into IBM's Pascal, but the Quadlink system doesn't permit the use of Apple CP/M disks, so other business and professional uses will probably be limited.

Other Choices for Storage

The three major reasons why people choose to buy storage devices which do not use standard Apple disks are to get more storage space per disk, to speed up the operation of their drive system, or to use a kind of storage medium which can be exchanged among computers from various manufacturers.

The only type of drive system which gets you big gains in all three of these categories are the eight-inch drives from Vista. Hard disks and the Vista V1200 Amlyn system can provide 50 to 100 times the storage of an Apple disk, but everything has to be chopped back down to Disk II size if you want to carry information from your machine to another computer.

Many of these systems can provide some improvement in the time it takes to load a large program, but this is usually a matter of a modest increase in convenience. However, if you make heavy use of database systems for sorts of large files, the right choice here can completely change the way you do your work.

Using dBase II with the 6 MHz PCPI card (see Chapter 30), a 100K file with 1000 entries takes about 45 minutes to sort with a Disk II. That time drops to 25 minutes with a Vista eight inch system and to about 11 minutes with a good hard disk. Using a Neptune Ramdisk can cut the time to about seven minutes, but a PCPI Ramdisk extender will do it in about 90 seconds. Inserting a record in the file with a Disk II can take 15 minutes, but it's done in 15

or 20 seconds with a PCPI Ramdisk system. This means that you could do a month's work in a few hours one afternoon, but what it really means is that you can undertake tasks which you otherwise never would have seriously considered attempting.

Increased Storage Capacity

The least expensive approach to increasing storage capacity is to get a drive which can be operated by the Disk II Interface Card. To review, a "track" that can be read by the Disk II Interface card holds about 4K bytes of data. The Disk II drive has 35 tracks (at a density of 48 tracks per inch or tpi) so you get 35 × 4 = 140K. In practice, most word processors maintain multiple copies of your text on the disk and you end up with a practical working maximum of about 40K for word processing text files, which is equal to about 25 pages of double spaced typed text.

If you want to use larger files, you need to have more tracks on your disk. Using the same 48 tpi technology, it is possible to build a drive which can use a full 40 tracks, and this provides about 160K, a modest increase. Another option is to put a second read/write head into the drive cabinet. One head scans 40 tracks on the top of the disk, the other head scans the bottom, and you get 80 tracks total.

For several years, it has been possible to build 5 1/4 inch floppy disk drives with a density of 96 tpi and several companies offer 70 or 80 track drives based on this technology. The 70 track drives provide about 290K of storage, while the 80 track systems bring this up to 320K. The tracks written by these drives are so thin, however, that disks written with one of these systems may be unreadable on your Disk II drive.

You can get 40 track, 160K drives from Microsci (the A40 for $449) and from Rana (the Elite 1 for $379). Only Microsci markets a 290K 70 track drive (the A70 $599). The Visa Duet is a double sided, 40 track drive with a total of 80 tracks, but Vista likes to sell two Duets side by side in a single box, which they call the Quartet. This gets you a total of 160 tracks for 640K ($849 for the pair). These drives can be operated in Disk II emulation mode with standard DOS where they act like two 35 track drives.

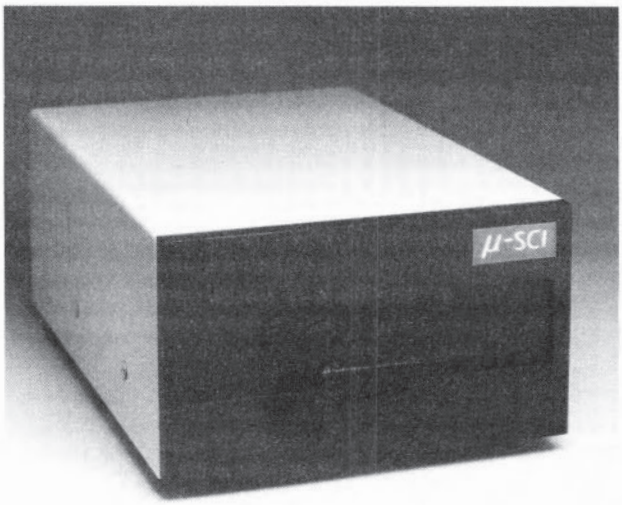


Fig. 24.5 MicroSci A82 high capacity 5¼ inch drive.

The reason that it is a good idea to buy high density drives in pairs is that if you have one high density drive and one Apple floppy, there's no way you can make a backup copy of a very large file without first breaking it up. MicroSci (A82; see Figure 24.5) and Rana (Elite 2) sell single sided 80 track drives providing 330K. The Rana Elite 2 (see Figure 24.6) goes for $649 or you can get the Elite 3 which is a double sided drive with 80 tracks on either side of the diskette (650K for $849). Although the Vista Quartet provides for easy back up, the Elite 3 performs a little bit better in terms of speed and accuracy of disk accesses.

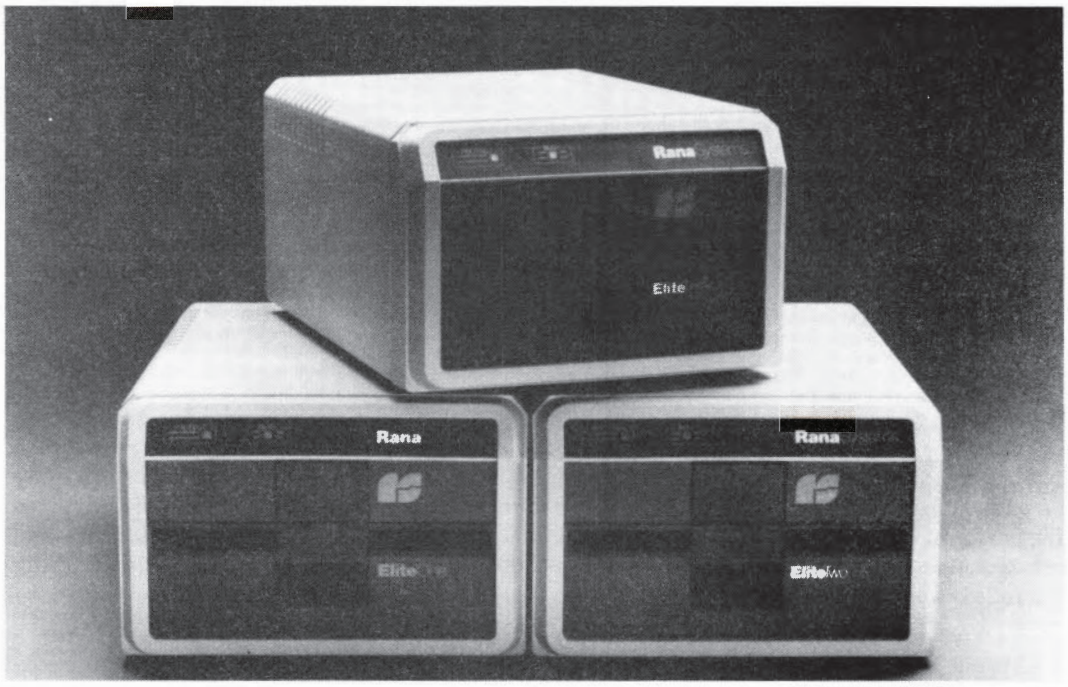


Fig. 24.6 The Rana ELite One, Two, and Three offer 160K, 320K, and 640K of storage.

One particularly interesting way to buy Rana Elite 2 type drives is with an IBM PC wrapped around them. This is, in effect, what the Rana 8086/2 (see Chapter 30) makes possible. It is a box which contains two slim line equivalents of the Elite 2 and which you can use from DOS 3.3, ProDOS and CP/M. Part of the interface card and the drive controller are built along standard Apple compatible lines. However, if you insert an IBM PC type disk (MFM instead of GCR; see Chapter 23) a different part of the controller card is activated, and several additional circuit boards within the box are activated. These boards include a complete PC compatible computer with 256K of RAM, an 8086, and an extended set of PC graphics modes.

Microfloppies

The least expensive high capacity drive is the Amdisk-1 from Amdek (see Figure 24.7). This drive actually uses three inch diskettes, but it plugs into the Disk II Interface Card and, from the controller card's point of view, looks as if it were a 5 1/4 inch drive. This kind of "microfloppy" diskette is based on advances in drive technology which permit recording at 135 tpi as well as an increase in the density within each track. The tracks are shorter, but they contain just as much information. The capacity which results from all this is 286K bytes, and the drive costs just $299. The three inch diskettes are sold by Amdek for $7 per diskette.

Fig. 24.7 Amdek's "Amdisk" 3 inch disk drives use a mechanism from Hitachi rather than the 3½ inch Sony mechanism in the Macintosh. Unlike the Sony drives, they provide good emulation of a standard Disk II.

There are several different kinds of sub-5 1/4 inch diskettes. Sony is producing a 3 1/2 inch diskettes, Dysan is backing 3 1/4 inch drive, and Hitachi (with Maxell) has been backing the three inch system used by Amdek. The market is sorting out rapidly in favor of the Sony 3 1/2 inch diskettes. One version of the Sony system spins the disks at 600 rpm. This improves the performance of the drive, but makes it incompatible with 5 1/4 inch controllers and software. The most popular version is spun at 300 rpm and is fully compatible with existing controllers and software.

The Apple Macintosh uses a Sony drive with variable speed (see Chapter 23), which is controlled by the same IWM chip used in the //c. The diskette is packaged inside a rigid plastic cassette and has a spring loaded sliding door to keep dust out when the cassette is not in the drive

Double Density Controllers for Very High Capacity

A different approach to increasing the storage capacity is to increase the information stored in each track using a process called "MFM recording" (see Chapter 23). This can't be done with a Disk II interface card, so you have to buy both a high density controller and a high density drive. Buying a completely different controller card means that you must be very careful to make sure that there is adequate software support. Further, data will be read off the disk at twice the speed, and this means that DOS may be overpowered by the rate of information flow. Fortunately, the one popular high density controller, the Vista A800/801, has excellent software support, and uses DMA (see Chapter 27) to provide an enormous increase in access speed over standard Apple DOS. Some programs will load 20 times as fast as with a Disk II.

The Vista A800/801 controller costs $379 before you even buy the drive, so it doesn't make sense to buy it unless you're going to get a really substantial amount of increased storage. Once again, the situation is quite good. The Vista A801 is sold with a six megabyte floppy

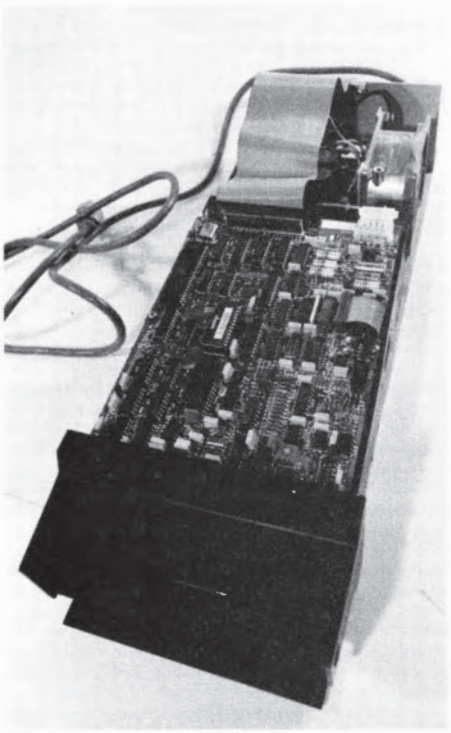


Fig. 24.8 The Vista V1200 uses a cartridge with five 1.2 megabyte floppies.

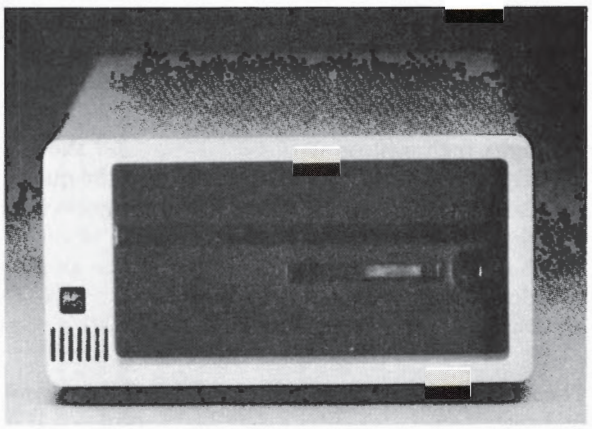


Fig. 24.9 The 2.5 megabyte floppy drive from Rana. This double-sided drive offers twice as much storage per disk as the V1200, and is more reliable because it does not use a mechanical disk picker.

system called the V1200 (see Figure 24.8). This system has just one read/write head, but it works at a density of 170 tpi and uses the A801 to get MFM double density in each track. This results in a storage capacity of 1.2 megabytes on one side of a special 5 1/4 inch floppy.

The V1200 also includes a "disk picker" mechanism which selects any one of five floppies packaged together in a cassette. Although only 1.2 megabytes is on line at any one time, the V1200 can move to another platter in about two seconds. It is possible to remove diskettes from the cassette, which is nice for backup. You can copy a one megabyte file from one platter to another, but this requires a great deal of disk picking and is a bit slow. This is a wonderful system for backing up files from a hard disk, but it is also a very economical choice for a high density floppy system. The controller and drive list for $1,550, and each six megabyte cartridge costs $50. This means 10 times the storage of a Rana Elite 3 for twice the cost. Note, however, that the V1200 uses very new technology and that always means you're taking a few chances.

The 2.5 megabyte 5 1/4 inch floppy from Rana (see Figure 24.9) uses a similar technology but writes on both sides of a disk. It uses just one disk at a time so there is no "disk picker" to go awry as in the V1200.

Eight Inch Drives for Compatibility and Capacity

The Vista A800 controller is used to operate standard eight inch double sided double density drives such as the Shugart 851. Eight inch disk drives are extremely rugged and reliable and today's drives reflect 10 to 12 years of experience in large scale manufacturing. The capacity

of a Shugart 851 is 1.2 megabytes, and the drive itself is similar in cost to a Disk II. However, to set up a complete eight inch drive system, you need a cabinet, a power supply, a fan and some cables.

An eight inch system with two drives and the A800 controller can be bought for anywhere between $1,200 and $2,500 depending on the quality of the components you choose. Vista sells several different eight inch system configurations. The most popular includes their V1000 cabinet which is nicely designed table top or rack mount arrangement with a very quiet fan. You have your choice of installing just one single sided single density drive, on up through two double sided double density Shugart 851's ($1,800). Vista also sells a half height dual eight inch system called the V1100 ($2,200).

These drives use the standard IBM 3470 or System 34 format, so you can freely exchange disks with an S-100, Xerox, similarly equipped IBM PC, or just about any other CP/M microcomputer which can use eight inch disk drives. Within the Apple environment, the A800/801 is compatible with CP/M from Microsoft and PCPI, MS-DOS from the ALF 8088 card (see Chapter 30), Pascal 1.1, Pascal IV.0 and Pascal IV.1. Many good programmers own this card, so there are even utilities and copy programs which are traded around among A800 owners. Note, however, that the A800 is a very high performance card which makes strong demands on the Apple. Some older Apples and some early Apple //e's (revision A) don't do too well when the A800 is used in conjunction with another DMA device such as the Microsoft Z-80. The PCPI Z-80B, which does not use DMA, is a better partner for the A800 in the //e.

The eight inch drive systems from Lobo and Taurus can't match Vista's software support, but these companies may be able to offer a better price. Sorrento Valley Associates makes single density controller cards for eight inch drives, but it is difficult to justify the cost of the controller because the storage capacity of the resulting drive system is not very great.

Laing Electronics assembles eight inch systems based on the Vista A800, and often can provide better prices than Vista without sacrificing on the software. And then, of course, the hobbyists in the crowd can always buy an A800 and then trundle down to their local computer junkyard, pick up a couple of aged "Shugies" and forge together a two megabyte floppy system for less than $1000.

Comparison Shopping for Hard Disk Drives

Contrary to popular belief, not all hard disk systems are alike. If you read the detailed "spec sheets" for any hard disk system, all of them have exactly the same numbers. This is because nearly all of them use the Seagate 406/412/419 series drives, and if they don't use Seagate drives then they use drives from some other company which claims full Seagate compatibility.

The actual drives may all be the same, but the drive is just one of four parts of a hard disk system. The second part is a controller card (see Figure 24.10) which directs all accesses to the information tracks, manages formatting, and is responsible for error detection and correction. The third part is a "host computer adapter" which supervises communication between the Apple's motherboard and the controller card and determines most of the speed and performance characteristics of the system. The fourth part is the software which operates the other three parts.

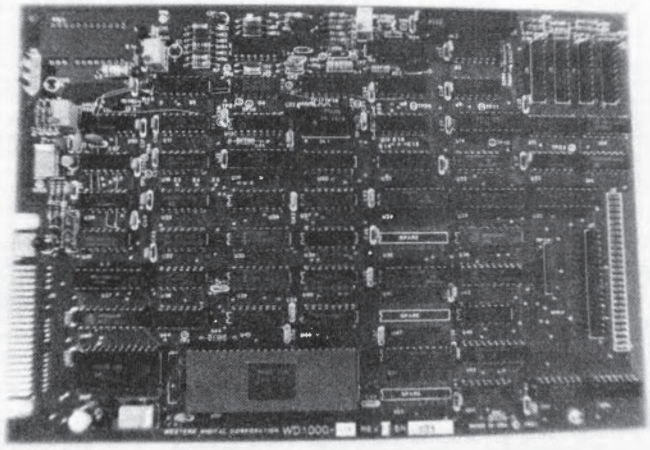


Fig. 24.10 A hard disk control system usually involves a "host computer adapter" in the Apple slot and a generalized controller, such as this WD-1000, in the drive cabinet. The data rate of 5 million bits per second describes only the transfer from the disk surface to the buffers on the controller. Transfer from the controller to the host computer adapter is far slower.

Reliability of the Drive Itself

The Seagate drive hardware is discussed in several places in Chapter 23. For the most part, if you buy a five, 10 or 15 megabyte 5 1/4 inch drive, all you need to worry about is whether or not it's a real Seagate. Hard disks are assembled in dust free "clean rooms" by people wearing masks and gloves, and the drives should be thoroughly tested on an individual basis. Nonetheless, hard disks do "crash" occasionally with fairly disastrous results, and your best protection is to be sure you get a Seagate, Miniscribe or Quantum mechanism in the first place.

Percom, for instance, uses a Seagate look-alike from CMI, but the CMI manual is quite frank; all of the specifications listed are identical to the Seagate except that they say that their drive is about 30 percent more likely to fail than a Seagate. This probably saves the buyer $200 to $300 at initial purchase, but you probably pay dearly later on. The code words are "MTBF: 8,000 POH" which you probably would never guess means "mean time between failure is about 8,000 hours of use."

Flexibility of the Formatting System

Formatting a hard disk is a tedious process which can take over half an hour. Since many people buy hard disks in order to speed up their work, formatting is not something you want to do very often. It would be nice if you could format the drive when it first arrived and never go through the process again. However, many Apple users do some of their work in DOS, some in Pascal, and some in CP/M, and each of these operating systems requires a different kind of formatting. On most of the popular hard disk drives it is possible to "partition" the drive into three regions, each formatted for one of the three operating systems.

If you decide at the outset to divide the space equally between each of the three operating systems, your five megabyte drive will force you to work in a 1.6 megabyte space at any one given time, and that's less storage than two boxes of Apple floppies. If you also use ProDOS, you'll probably need four areas. Corvus makes the situation a bit simpler since its drives cannot work with CP/M at all. It does force you to make an early decision about how much will go for Pascal and how much for DOS, and you have to erase everything and reformat if you guessed wrong. In fact, the drives from Corona, Corvus, Davong, Mitac, Percom and Xebec all require reformatting if you want to rearrange the partition.

This "pre-allocation" requirement also applies to the organization of information within the operating system partition. Information on a hard disk is stored in "volumes" which are the logical equivalent of individual floppy disks. If a volume is too large, it can get hard to read the directory. If it is too small, some file might not fit. If you chose to set up 200K volumes at the time you bought your disk, than you're going to be just plain stuck when your data base reaches 201K; it's too big to copy onto an Apple floppy, and you'll have to erase and reformat the entire drive if you want to make the volume larger.

Alone among Apple hard disk manufacturers, Mountain Computer Products offers "Dynamic Volume Allocation." This means that you can rearrange your volume sizes as you work in order to fit your current needs. CP/M users who buy the Mountain hard disk can designate eight different volumes as, for instance, drives C:, D:, E:, etc., a process called "mounting" the volume. The Corona Starfire also allows eight active CP/M volumes, but Percom, Mitac and Xebec permit just two, and Corvus allows none.

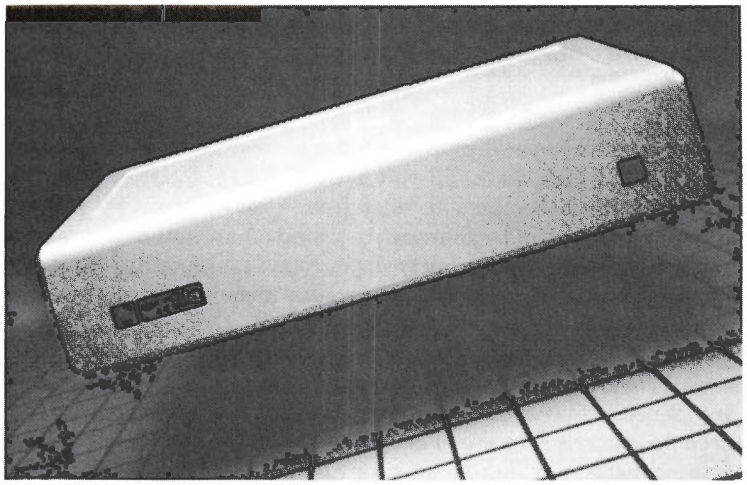


Fig. 24.11 Profile hard disk.

ProDOS compatibility is one other concern to keep in mind. There is nothing very difficult in designing drive control software to work with ProDOS, but many of the current systems will have to be rewritten. One company with a slight edge is Apple itself, since the Profile (see Figure 24.11) for the Apple II is sure to have proper ROMs on the interface for ProDOS. Quark's hard disk for the //c is also a good choice for ProDOS software since Quark also publishes Word Juggler and several other ProDOS-based software packages.

The Speed Bottleneck

The major determinant of the speed of the system is how fast the "host computer adapter" card can hand data across to the modified operating system. All of the drive manufacturers advertise a "Data Transfer Rate" of five million bits per second, but that is completely meaningless. It actually does describe the speed at which the drive sends data to the controller card, but the speed at which it gets from the controller card into the Apple is all that you care about and that speed is over 1000 times slower.

The reasons for this speed bottleneck are described in Chapter 23. The only way around it is to build a fairly elaborate host computer adapter card and write some very good operating system software. None of the hard disk manufacturers have done this. Corona and Mountain each include large storage buffers to capture incoming data, and this greatly improves the

efficiency of the transfers, but the hard fact is that the Vista A800 controller for eight inch floppy disks loads data faster than most hard disk drives.

The reason for this bizarre situation is that most manufacturers of hard disk systems are interested in selling to a broad variety of people with different kinds of microcomputers. They can use the same drive and the same controller for all computers, but must design a host computer adapter and software for each different microcomputer. The drives come from Seagate, the controllers come from Xebec or Western Digital, and the companies you actually deal with may know very little about Apples, as well as very little about coaxing performance out of a controller.

The entry of Mountain Computer into the Apple hard disk market, however, bodes well for the future. This company has been producing high performance Apple peripherals for nearly five years and certainly has the ability to design a good Apple interface. Their dynamic volume allocation system already makes the Mountain hard disk the one outstanding hard disk drive currently available for the Apple.

The other major new current in Apple hard disks is the entry of Apple itself. ProDOS permits the use of Apple's ProFile hard disk drive. This is a system that uses an older Seagate drive called the ST 506. ProDOS is very well suited to managing huge numbers of files on a hard disk and does not require any preallocation of volumes.

Hard Disk in Networks

A single high capacity hard disk can provide enough space for several microcomputers, so Davong, Percom and Corvus all offer means of connecting their drives into a "network" which links several computers. The various types of networks are discussed in Chapter 20. The Corvus network is Omninet, while both Percom and Davong use Arcnet style systems. The Davong system permits one large drive (see Figure 24.12) to run all of MS-DOS, CP/M 86, CP/M 80, DOS 3.3, and Pascal, so it can be used by IBM PCs, Apples and a variety of other microcomputers. The Davong "Multi-OS" network system is quite close to full OSI architecture and has a very thorough software base.

Fig. 24.12 Davong hard disk.

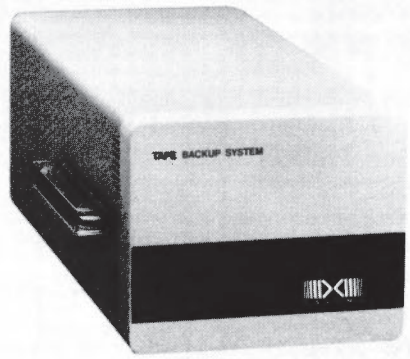


Fig. 24.13 Davong tape backup system.

Hard Disk Backup

A hard disk "crash" is approximately equivalent to someone dropping a load of bricks onto your drawerful of 100 Apple floppies, instantly and completely destroying all of your program and data files. And hard disks do crash. During transport, the disk's read/write head is stowed over a special transport track, and when the drive is not running, the head is usually sitting on a "landing pad" within the drive.

However, when the drive is running, the head skims along over the surface of the disk at a height of 20 thousandths of an inch, supported only by a cushion of air. Many manufacturers warn that an "acceleration of 1G" can crash the disk. This means that if you hold a running drive about one inch above a flat surface and let go, the head will crash and destroy the disk the instant the drive begins to fall and no additional damage will be done by the actual impact.

Don't get anxious, make backups.

The problem with backing up a five or 10 megabyte hard disk has to do with the question of where to put all that data. Presumably, you bought the drive in order to have large files; but if your only backup device is an Apple floppy, none of your files can be much larger than they were before you bought the hard disk. If you happen to have an older Video Cassette Recorder (VCR) around the office which doesn't have color correction circuitry, then you can just plug a Corvus drive into the VCR and the Corvus "Mirror" system will automatically copy the entire contents of the hard disk onto the cassette and reload it later if desired. Otherwise, you're going to have to consider purchasing some sort of alternative backup system.

A streaming tape system (see Figure 24.13) can copy a 10 megabyte disk in one or two minutes, but these systems can cost as much as the hard disk. One new option is buy a V1200 six megabyte floppy system from Vista. By itself this a very formidable disk storage device, but since it can accept high speed transfers of up to 1.2 megabytes on each of its five platters, it also makes an ideal and competitively priced hard disk backup system.

Although removable hard disks from such companies as Axlon and Digital Electronic Systems are often advertised as principal storage media, many of these devices are actually being used for backup of fixed disks. The successful operation of a removable hard disk depends on being able to lift the read/write head mechanism off the disk surface and then opening a cassette to actually get the disk completely free for removal. There is substantial risk of getting dust or even cigarette smoke into the cassette during this process, thus leading to a crash. As a backup medium, however, such a crash is not really any sort of disaster because the cassettes themselves are not extremely expensive.

Another important new consideration in hard disk backup has to do with ProDOS. The operating system will automatically "flag" each file that you use as you use it. At the end of the day, these ProDOS "use flags" can signal which files need to be backed up. As they are copied, ProDOS "clears" the flags to prepare for the next round of use. This way, you can avoid both the waste of recopying the whole five or 10 megabytes every time you do backup, and you can also avoid the tedium of sitting there at the end of the day, scanning directories to remember which files need to be backed up.

Nine Track Magnetic Tape

A standard 10 1/2 inch magnetic tape can store 40 megabytes of data and can stream the information into an Apple at over 70K bytes per second, which is faster than most hard disks. However, these system can take a great deal of time to access a file before it can be read in.

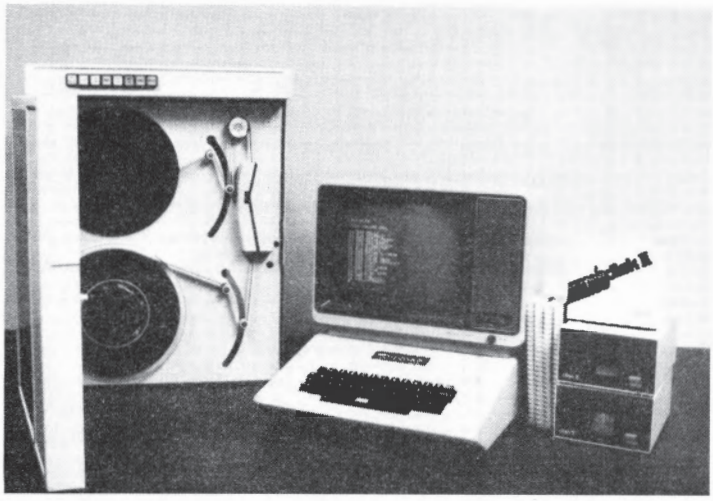


Fig. 24.14 Nine-track mag tape can be used for bulk transfer of large amounts of data between micro and mainframe. IDT sells tape transports as well as controllers.

Magnetic tape has an enormous advantage over a hard disk in that you can take your 40 megabytes out of the "tape transport," carry it over to the local mainframe computer (DEC, IBM, Control Data, etc.), and continue your work. This is also a simple means of gaining access to several commercially available databases and government data files which are distributed only on magnetic tape.

On a magnetic tape, data is stored as parallel bytes with a ninth parity bit for each byte. The read/write head has nine elements side by side and they lay out nine tracks along the tape. Block size is variable and can range from 256 bytes up to 32K bytes of uninterrupted data. The bytes are stored at a density of 800 bytes per inch with Non Return to Zero Inverted (NRZI) or at a density of 1,600 bytes per inch Phase Encoded (P.E.).

Two companies sell complete nine track magnetic tape systems for the Apple. Electrovalue Industrial provides a controller and software for $900. The controller plugs directly into an Apple slot and can be connected to a variety of commercial drives. This controller is only capable of writing and reading NRZI data (800 bpi). Electrovalue also sells the Pertec 7000 tape transport which operates at 18.75 inches per second and can only manage seven inch tapes ($900), and the Wangco/Perkin Elmer Model 1025 which handles full sized 10 1/2 inch tapes and runs at 25 inches per second ($2,100).

A more versatile and more expensive system is available from Innovative Data Technology (see Figure 24.14). These folks use a very elaborate external controller (series 2600 formatter/controller) which connects to the Apple via an RS-232C or IEEE-488 interface. Some models of the controller can be provided with a 32K RAM buffer, so it is fairly convenient to work with full-sized blocks. Commands and data are sent to the controller along these standard interfaces, so it has been a simple task for IDT to manufacture interfaces for IBM PC, Apple II and III, TRS-80, PET and other microcomputer, minicomputer and mainframe systems.

IDT assembles their own high performance tape transports. The TD 1012 series has a streaming tape mode of 100 inches per second and a read mode of 12.5 inches per second. These drives offer both 800 bpi NRZI and 1600 bpi P.E. recording. A complete system including the TD 1012 tape transport and formatter/controller costs $6,495. There is a higher performance model, the TD 1054, which reads at 45 inches per second and in which a 32K buffer as well as the controller/formatter are built into the drive cabinet ($8,500).

Disk Drives without Moving Parts

Looking back to Chapter 23, you will recall that the role a disk drive plays in a computer system has nothing fundamental to do with the fact that there is actually a spinning magnetic disk out there. Much of this part of the book has been devoted to an exploration of the 6502's address space, and the sections on drives have been treated as an exploration of "extra address spaces" which exist only in the mind of DOS or ProDOS. These extra address spaces are organized (in ProDOS) as an ordered, addressable set of 65,535 blocks, with each block containing 512 bytes of information. Each of these ProDOS address spaces can therefore manage 65,535 by 512 equals 32 megabytes of data, so the trend has been to put disk drives into these address spaces.

ProDOS, however, cares little about whether the bytes are spinning around on plastic disks or loaded into huge banks of RAM memory chips, as long as the appropriate 512 bytes shows up when it calls up a specific block. Apple floppy disks, high density floppies, and hard disks all have their special features, advantages and disadvantages, but there are other kinds of "block storage devices" which may be more appropriate for certain tasks. There are now more than 20 companies which sell various kinds of block storage devices with no moving parts.

Solid State "Disk Drives" and Bank Switched RAMdisks

All but one of these solid state block storage devices use RAM chips instead of disks to hold the data and the odd one out (MPC) uses a newer technology called a "magnetic bubble." The ones made from RAM chips are collectively called "RAMdisks" and MPCs can be called a "Bubdisk." Most of the RAMdisks can be used either as a bank switched addition to the 6502's memory space for use by running programs, or as a block storage device for DOS, ProDOS, CP/M or Pascal. These "bank switch RAMdisks" are discussed in Chapters 22, 25 and 26. The Bubdisk and the remainder of the RAMdisks can't be bank switched, and thus can only be used as "solid state disk drives."

In general, the bank switched RAMdisks generally offer greater speed and greater versatility than the solid state disk drives. In bank switching, a large range of RAM is temporarily moved directly into the 6502's address space. Some RAM disks can deliver nearly 48K in less than a microsecond (millionth of a second). A pure solid state disk drive that does not do bank switching must feed the bytes into the Apple one by one, usually at a rate of about 10 microseconds per byte. Further, when ProDOS calls for access to a different block, the solid state drives must provide a mechanism for rearranging the flow of data in response.

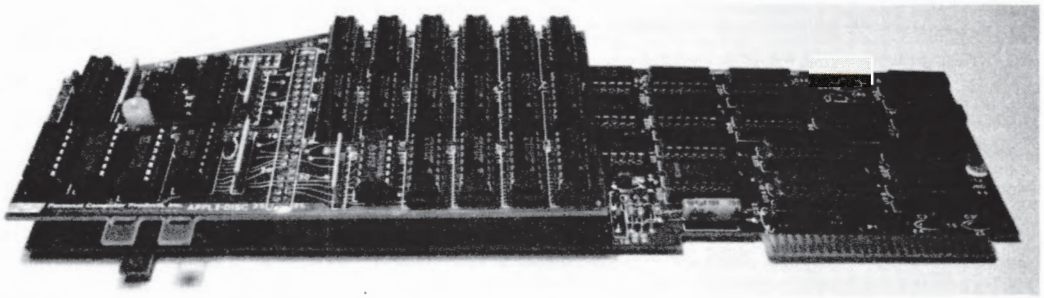


Fig. 24.15 The PCPI 128K Applidisk RAM extender mounts on the side of the PCPI Z-80B co-processor card.

All of these considerations aside, the real problem with speed is similar to the situation with hard disks. Delivery of the bytes to the file buffer may be quite rapid, but the operating system which must move the data out of the file buffer to its intended destination can be the real bottleneck.

Nearly all RAMdisks provide a speed increase of between two and 10 times over an Apple floppy, depending on the particular task. The only RAMdisk that is far faster than this is the PCPI RAMextender (see Figure 24.15). When operating with DOS and the 6502 it is similar in speed to the other RAMdisks, but when operating with the PCPI Z-80 board it provides a speed increase of nearly 30 times over an Apple floppy. When the same PCPI Z-80 is used with a standard RAMdisk this speed advantage disappears.

Advantages of Solid State Drives

Although these devices cannot be used by the Apple's main memory, some of them offer unique advantages that you cannot get from a bank switched RAMdisk. One of these has already been mentioned, and that is the spectacularly high speed of the PCPI RAMextender operating with the PCPI Z-80 coprocessor. The high speed of this system is due to three factors. The first is that the PCPI Z-80B co-processor itself operates at very high speed. However, this processing speed advantage is lost when any other RAMdisk is used because all other RAMdisks must use the Apple's 6502 to deliver each byte to the Z-80.

Second, the PCPI RAMextender plugs into the side of the Z-80 card so the bytes don't have to travel the complex path through the Apple. The third reason is that the designers of the hardware chose to hire one of the top machine language programmers in the country to write the operating system software. This choice is evident in a variety of ways (good eror handling, easy to use command menus, etc.), but a dBase II sort which runs in 90 seconds instead of 45 minutes is enough of a testimonial.

The Bubdisk for Use Near Jackhammers

One weakness of floppy disk drives (and fatal weakness of hard disk drives) is that their high speed, high precision mechanical parts are very sensitive to vibration, abrupt temperature changes, and the ravages of very young or very casual users (i.e., visitors to a display in Disneyland, etc.). The Apple itself is legendary as a rugged and nearly indestructible device, but since it's relatively useless without disk drives, it is usually relegated to clean quiet desk tops where it never gets a chance to show how tough it is.

Any RAMdisk is fairly immune to the mechanical disruptions which impair or destroy disk drives, but the problem is that since RAMdisks lose their information when the power is turned off, you still can't transport a RAMdisk-equipped Apple without bringing along a disk drive. That is, unless you have a Bubdisk rather than a RAMdisk.

The MPC Bubdisk (see Figure 24.16) uses a kind of storage device called Magnetic Bubble Memory (MBM). A single MBM chip such as the Intel 7110-4 used by MPC can store 128K bytes, and unlike a RAM chip it does not lose its memory when the power is turned off. An MBM chip is very different from a RAM chip and in fact from any other kind of electronic chip. It is not made of silicon and is not a semiconductor, but instead is made of a thin film of garnet and gallium and gandolinium sandwiched between two permanent magnets.

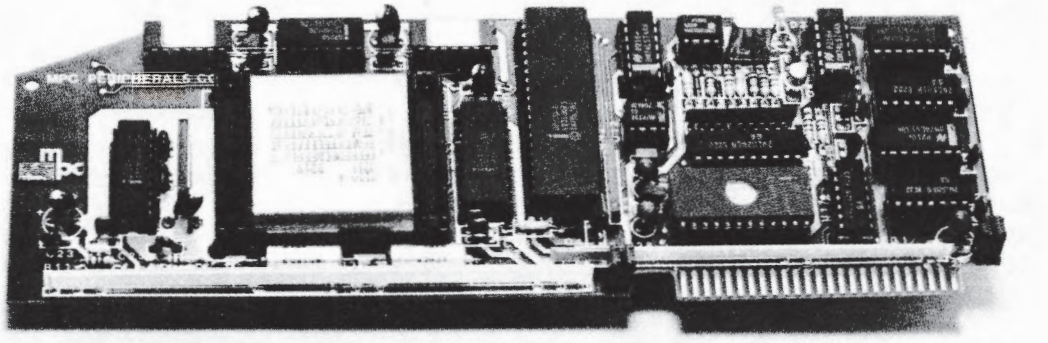


Fig. 24.16 MPC Bubdisk. This solid state drive is based on a 128K magnetic bubble memory device from Intel.

The garnet film normally contains randomly arranged magnetic domains, but the permanent magnets are used to organize these into discrete "magnetic bubbles" within the plane of the film. The rest of the chip is designed to marshal these bubbles into rows and columns and to actually move them around within the garnet in thousands of neat little loops for reading or writing. There is a fairly thorough description of the details of all this in *Handbook of Semiconductor and Bubble Memories* by Walter A. Triebel and Alfred Chu (Prentice Hall). The access time for bringing a particular set of bubbles over to the read sensors is just a little better than for a floppy, so this is not the best choice for high speed solid state disk operations.

In use, the MPC card is just plugged into an Apple slot and treated as if it were a disk drive. You can transfer programs and data into the Bubdisk, then turn the Apple off, remove the disk drives, and carry the whole thing off into a "hostile environment" for extended use.

Solid State Drives Made with RAM Chips

The solid state RAMdisk from Pion is built to look like a disk drive in that, like the Axlon Ramdisk 320 (see Chapter 23), the RAM chips are in a separate box outside the Apple. The Pion Interstellar Drive ($1,095) comes with 256K bytes, but can be expanded up to one megabyte at a cost of $595 for each additional 256K bytes. RAM chips can require a lot of electrical power, so Pion has arranged to have the device plug into the 120 Volt AC wall socket on its own. This is important because the Apple power supply is easily overtaxed. There is also provision for short term battery backup so you don't lose everything whenever you have to turn the Apple on and off or if there is a power failure.

The Pion system includes some fairly sophisticated circuitry which handles the detection of errors in transmission and the insulation of the RAM chips from electrical noise generated by the Apple when it is turned on or off. Like some hard disk controllers, Pion includes an error checking and correction system which involves storing "cyclical redundancy check" information along with the data.

The Pion box is actually an intelligent RAM device in that on board circuitry is responsible for selecting and presenting bytes to the Apple. This pertains to an important distinction to be made in evaluating the speed of data transfers in a RAM disk device. There are really two events involved in getting data from a RAM disk to a file buffer. The first is getting "visibility" for the RAM i.e., making it possible for the 6502 to address it. The second is the actual transfer.

When you do bank switching of large amounts of RAM, the rate at which large numbers of bytes gain visibility is very high; in a few microseconds, 16,000 bytes pop into the 6502's line of view. Then the 6502 must go about moving the bytes. To do this, it chooses the starting point of its read and the starting point of its destination and begins incrementing along through both. The cycle is index the read location, get the byte, index the write location, store it, etc.

In the Pion system, only one byte is presented at a time, but it is always presented at the same location of the single data port on the interface card. Therefore, the transfer cycle doesn't require an index of the read location: get the byte, index the write, store it, etc. Thus the transfer cycle is faster with the Pion system than with a bank switched system. So although the visibility rate is extremely fast for a bank switched system, the real bottleneck is in the transfer.

The solid state RAMdisks from Sorrento Valley Associates and from Synetix (see Figure 24.17; 147K for $395 and 294K for $695) are complete on a single board and plug directly into one slot in the Apple. However, they don't offer battery backup, do draw a fair amount of power, and can't be used as direct memory extensions by the 6502.

It might have been possible for these companies to make their boards look exactly like a real Disk II from the point of view of DOS and RWTS. However, this would have pushed the price up and cut down on the available space for RAM chips on the board. As a result, you don't even get to use protected commercial DOS software. The Synetix and SVA systems are OK for use with CP/M, and may yet be adapted for full use with ProDOS commercial software. They do offer more RAM than similarly priced "bank switch" RAMdisks, but you lose a great deal of performance to get this improvement in price.

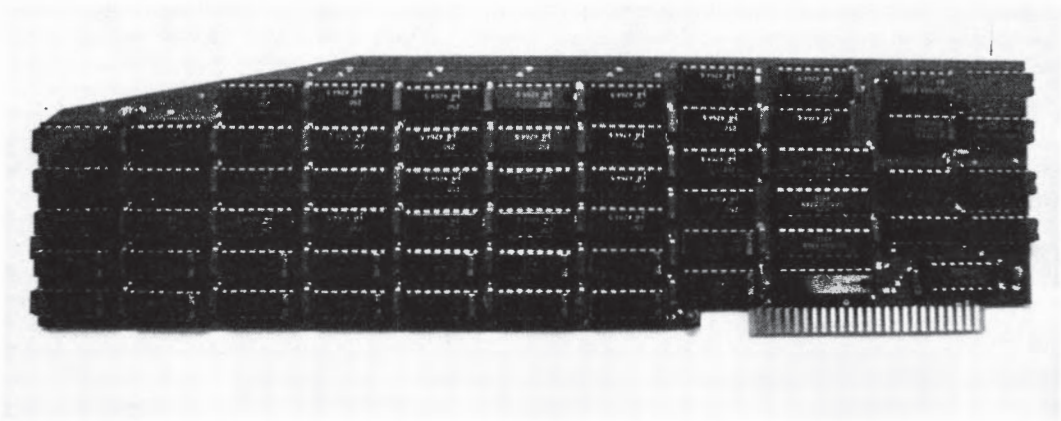
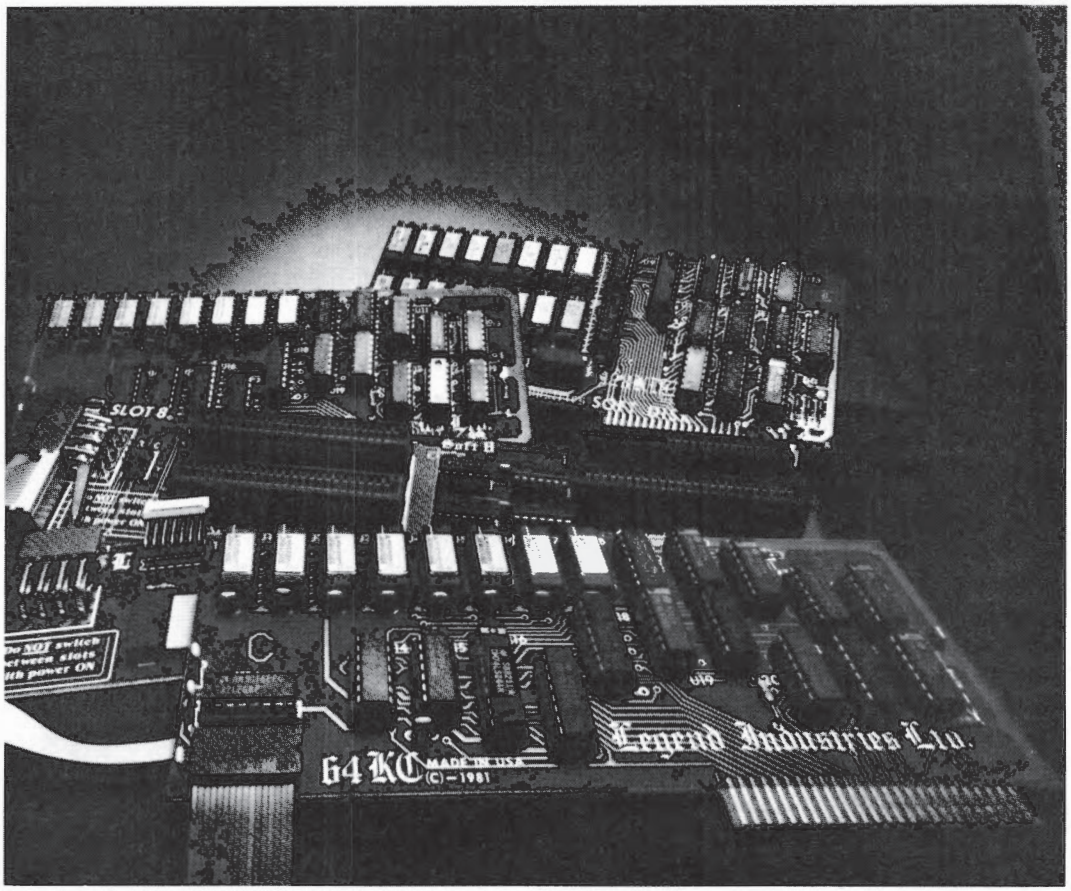


Fig. 24.17 Flashcard from Synetix.


SLOT 8
64 KC
(C) – 1981
Legend Industries Ltd.

Chapter 25

Bank Switching the High 16K

Bank Switches in the $C0 Page

Before proceeding, it might be helpful to take a quick look back to Chapter 21, Figure 21.4 and refresh your memory about the general layout of the 6502's address space in the Apple. In Chapter 21 we swept upward from $0000 to just below $C000 (49,152) thus covering the first 48K of RAM. Next we jumped to the very top at $FFFF (65,535) and worked down through the 12K of ROM chips that contain the Apple's Monitor and the Applesoft BASIC Interpreter until we got down to $D000 (53,248).

The final remaining 4K of addresses between $C000 and $D000 are covered in Chapter 22. To review, Chapter 22, Figure 22.1a, the top 2K of this $C000 space ("see thousand" space), is called the expansion ROM space and provides a shared area into which any peripheral card can switch its own 2K ROM chip. The switching of these Expansion or "$C800" ROMs is managed by several control lines as explained in Chapter 22. In the //c, there is no switching in this space since everything in the area is in permanent ROM.

In the space just below this, each peripheral card is assigned one page (256 bytes) of Peripheral Card or "$CN00" ROM address space. Each slot (except slot 0) has its own exclusive page so there's no need to do any switching. The Axlon system, Microtek BAM 128 and Q-Disk card, however, do their own on-card switching to bring any one of 512 extra pages of RAM into the space and thus implement one type of "bank switched RAMdisk."

The last range of Apple memory remaining to be discussed consists of the 256 bytes of the $C0 page itself. This page rarely contains any RAM or ROM. The locations are used as "switches" to operate external devices, to cause those devices to dump data or status reports into the Apple, and to cause those devices to grab data off of the Apple's data bus.

These "switch plate" areas are clumped together in groups of 16. The first eight "switch plate groups" are formally called the Peripheral Card I/O or "$C0n0" spaces, and the last eight are called the On Board I/O spaces. In the //e, slot 0 has disappeared and its switch plate can be included with the "built-in" set, and in the //c, all active locations in the $C0 page are built in.

So far, we have paused to look in detail at the switch plate assigned to slot 6 (see Chapter 22, Figure 22.1a, and Chapter 23) since this is the nearly permanent and ubiquitous home of the Disk II Interface Card. The next one to get close scrutiny will be the switch plate assigned to slot 0, in addresses $C080 through $C08F (49,280 through 49,295). This set of addresses is configured identically in most 64K Apple II/II+s, and in all //es and //cs.

The D-E-F Bank Switches

The set of switches beginning at $C080 are used to force the Apple's Monitor and Applesoft ROMs out of the highest 12K of the 6502's address space and to bring in RAM instead. Since we're talking about the D thousand, E thousand, and F thousand spaces (see Figure 25.1), it will be helpful to call these switches the "D-E-F bank switches."

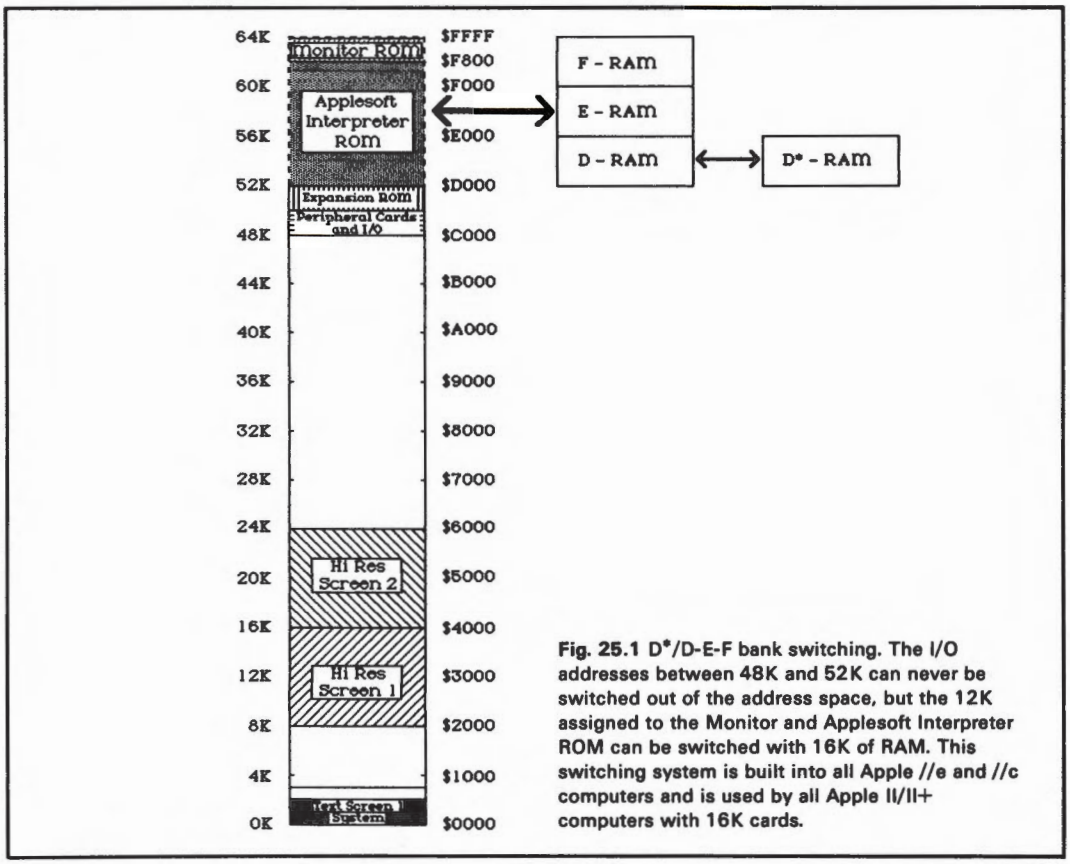


Fig. 25.1 D*/D-E-F bank switching. The I/O addresses between 48K and 52K can never be switched out of the address space, but the 12K assigned to the Monitor and Applesoft Interpreter ROM can be switched with 16K of RAM. This switching system is built into all Apple //e and //c computers and is used by all Apple II/II+ computers with 16K cards.

In the Apple //e, this RAM comes from the row of 64K RAM chips along the front of the motherboard, which also provides RAM for the first 48K of the 6502's address space. In the //c, it is in the first eight RAM chips towards the front of the machine (see Chapter 1, Figure 1.4). When the Apple is first turned on, only the first 48K of RAM locations is in the address space, and the switches in the $C080 group must be used to let the 6502 have access to the remainder of the storage cells in those chips.

So far, this means that you can actually use 48K plus 12K equals 60K out of your first 64K of RAM chips. Having 4,096 storage locations just sitting there unused seems wasteful, so some additional bank switches are included which can remove the 4K of RAM that had just been switched into the D Thousand space and replace it with that last 4K of memory cells. Thus we have 4K of RAM assigned to the F Thousand space, 4K assigned to the E Thousand space, and two ranges of 4K of RAM both of which are assigned to the D Thousand space. In order to get a little clarity, we'll call the first 4K that gets switched in the "D Bank" and the one which can then replace it the "D* Bank" (the Dee Star Bank; see Figure 25.1).

What Happens When You "Switch" a Bank of RAM or ROM

The case of the D ROM, the D Bank RAM, and the D* Bank RAM provides a good opportunity to pause for a moment and say something about what "bank switching" actually does. For purposes of explanation, it has been handy to talk about an "address space" defined by the 65,535 locations which can be described by the 6502's 16 bit address buffer. The convenient mental abbreviation is of a real physical space which various chips get moved in and out of. Of course, if you peek in under the hood, you'll see that nothing actually moves on the motherboard.

In the //e and the //c, the D space switching is actually done in two different ways. The first approach is to connect a ROM chip and a RAM chip to what amounts to the same address lines. When the 6502 announces an address in the $D000 range, one byte in the ROM chip and one byte in the RAM are identified simultaneously. Both have the same address.

Conflict is avoided by controlling which of the two will actually respond to the address. A "response" would be defined as grabbing a byte from the data bus in a "write," or dumping a byte onto the data bus in a "read." In this mode, then, we can better define "bank switching" as a means of controlling the "data bus response" of two memory devices which both have the same address.

Control Lines for ROM and RAM

Controlling the response of a ROM chip is very straightforward, because all ROM chips require an enable signal to operate. The detailed chores are handled by the Apple's Memory Management Unit (MMU). When the MMU wants information from the Applesoft ROM, it sends out an enable signal along with the address, and this permits the ROM chip to respond by dumping a byte onto the data bus.

At the same time, however, it is important to be sure that the byte of RAM with the same address does not respond. RAM chips do not have an enable signal (see Chapter 21, Figure 21.10), so the control problem is a little trickier. If you refer back to Chapter 21, you will recall that a 64K RAM chip has just eight pins for receiving a 16 bit address, so the address is sent in two waves. First the eight bit "row address" gets put onto the "RA address bus" and the Apple's timing circuitry (the PAL in a //e, the TMG in the //c) sends a signal called Row Address Strobe (RAS). This causes the RAM chip to seize the row address. Next, the MMU forwards the remaining eight bits of the full address onto the same "RA address bus," but this time the PAL sends a Column Address Strobe (CAS). The CAS signal causes the RAM chip to grab the rest of the address, pick out the appropriate bit of information, and dump the data onto the data bus.

When the MMU wants to disable the RAM chips on the motherboard, it fiddles with this CAS signal. It's a little trickier, in fact, since what actually has to happen is for the MMU to tell the PAL not to send the CAS. Because the column address strobe never arrives, the RAM chip never finishes its cycle, so there's never a data bus response.

One cute additional trick the MMU can play is to do all its reading from one bank, but all its writing to the other bank. The MMU just watches the read/write signal coming out of the 6502, and when it's a read, it enables the ROM and kills CAS. If it's a write, it turns on CAS,

but doesn't send its ROM enable signal. This provides a simple way of copying a range of information from one bank to another (read address, write address, move to the next address, read address, etc.).

Address Modification for the D* Bank

The MMU has a second and completely different way of bank switching which it must use when it switches between the D Bank and the D* Bank on the motherboard. Obviously, the ROM enable signal will be of no help, and it can't use CAS since that would kill the data response of both the D Bank and the D* Bank.

To do the D/D* switching, the MMU actually recalculates the address coming in from the 6502. When it is supposed to use the D* bank, and it receives an address between $D000 and $E000, it internally changes the address into one between $C000 and $D000 and then goes about two stepping it onto the "RA address bus." What about all that Peripheral Card ROM and Expansion ROM which actually have addresses between $C000 and $D000? Well, those guys never hear about it, because they only listen to the main address bus, and they have no idea what the MMU is actually putting onto its separate "RA address bus"—and could care less.

Operating the D-E-F and the D*/D Bank Switches

To sum up then, the three actual tools used for bank switching the high 16K in the //e and //c are the ROM enable signal, the CAS signal for motherboard RAM, and address modification by the MMU to get to the D* Bank. In the Apple II/II+, these banks are switched in almost exactly the same way, except that the signals are controlled directly by the switches in one of the $C0n0 "switch plates" on the RAM card. In the //e the same switches are used, but there is an intervening layer in that the switches themselves simply serve as signals to the MMU instructing it to actually take the necessary actions. The MMU is needed in the //e and //c because the D-E-F and D* bank switching system is just one part of a much more extensive bank switching arrangement (see Chapter 26).

In the course of working our way down through the Apple's memory, we've been looking at progressively smaller and smaller ranges. To understand D*/D-E-F Bank Switching, it won't do to look at whole pages, clumps of 16 bytes are too big, and for the first and only time we have to actually get down to particular individual bits within the Apple's address space. If you've ever looked at an explanation of these switches at the byte level (see, for instance, Table 4-5 in the //e Reference Manual), you'll be confronted by a bizarre series of assymetrical repeating address patterns. It's really very simple, as long as you just look at the bits.

The reason it was done at such an elemental level was that the original 16K RAM cards for the II/II+ were fairly expensive and it was important to simplify design. Further, it was intended that these cards be put in slot 0, and as you can see from Chapter 22, Figure 22.1a, there is no peripheral card ROM space for slot 0, hence control had to be very simple.

Assuming that the card is in slot 0 of a II/II+ or that we're talking about a //e or a //c, all of the switches are at addresses beginning with $C080. The bit pattern representation of $C080 is:

```
  C      0      8      0
1100   0000   1000   0000
```

As you will recall from Chapter 22, whenever an address between $C080 and $C08F is announced, the Apple II/II+ sends a signal to slot 0 called "Device Select" which might be called the "$C08x" signal (an internal signal in the Memory Management Unit of the //e and //c. If the card in slot 0 receives this $C08x signal, then all it has to do is watch the last four bits of the address.

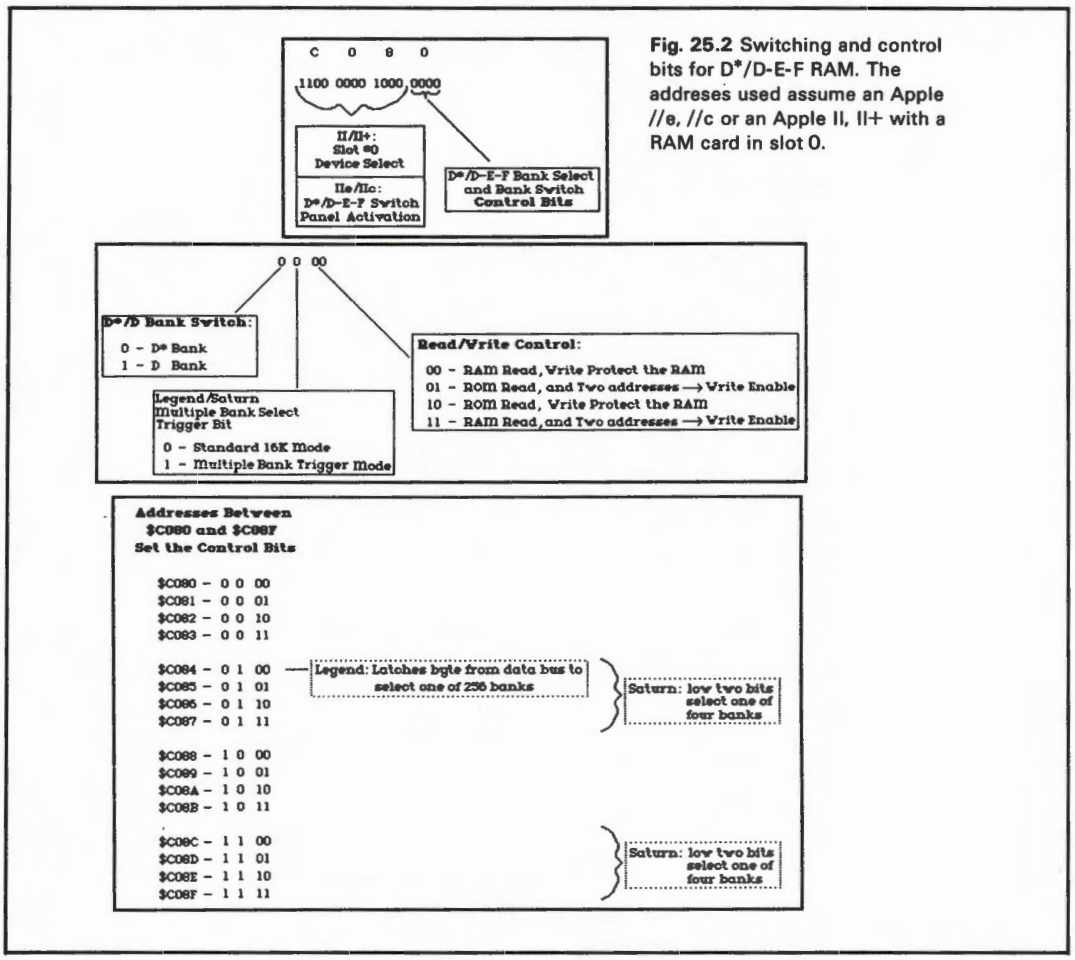


Fig. 25.2 Switching and control bits for D*/D-E-F RAM. The addreses used assume an Apple //e, //c or an Apple II, II+ with a RAM card in slot 0.

The bank switching system is based on the contents of these last four bits (A3, A2, A1, and A0). Some of the various possible bit patterns are laid out in Figure 25.2. In the scheme, the A3 bit operates the address modification system which selects either the D Bank or the D* Bank. If it is set to "one," then the remainder of the signals act on the D Bank; if it is set to "zero," then everything pertains to the D* Bank. This bit has no effect on the E thousand or the F thousand RAM.

The A2 bit is ignored by 16K RAM cards and by the //e. As we will see shortly, it is used in more elaborate memory cards to trigger the selection of one range of 16K of RAM from within a much larger array.

Operating the ROM Enable and CAS Lines

Most of the action takes place around bits A1 and A0. These bits are used to control the "ROM enable" signal, the "CAS enable" line, and to determine whether or not CAS will change when the 6502 goes from "read" to "write."

The actual bit arrangements and their results are laid out in Figure 25.2. The two bits are used to operate several arrangements of "Exclusive OR", AND, and other gates all of which is responsible for the odd arrangement. The "RAM Read" selection causes the 16K card to generate its own CAS signal (sometimes called "MCAS" just to keep things straight) and also to put a signal called "INH" (inhibit ROM) onto the peripheral card connector. In an Apple II/II+, the INH line is connected directly to the enable pins on all of the ROM chips. Thus, the ROMs will not respond to the address, but the RAM chips, supplied with a CAS signal, will.

In the //e, the MMU can disable the ROMs directly, but the //e was also designed to permit the use of a standard D/D*-E-F bank switch RAM card in any slot from 1 to 7. The additional problem in a //e is that if you turn on 16K in a RAM card, you not only have to get rid of the ROM that is in the space, but you also have to get rid of the high 16K of RAM on the motherboard.

To make this possible, the INH signal has two effects in a //e. It tells the MMU to turn off the ROM enable (ROMEN) signals as in the II/II+, but it also causes the MMU to have the motherboard CAS turned off. The proper name for the INH line in a //e might therefore be "inhibit motherboard memory." This only applies if you've added a bank switch RAM card in slots 1 through 7. Operating the built-in //e switches at $C08x has no effect on the INH line. Since you can't add RAM cards to a //c, the INH line disappears, and its input pin on the MMU is permanently held off.

By judicious use of A1 and A0 you can cause the 6502's "Read/Write" signal to reshuffle the switches which operate MCAS and INH. This results in the "RAM write protect" and "RAM write enable" features (see Figure 25.2).

16K RAM Cards for the II/II+

The basic 16K RAM card is one of the all-time most popular cards for a manufacturer to decide to make, and since you must have one installed if you want to use Apple Pascal or ProDOS, because these days they really don't cost too much, you should give this serious consideration.

Apple makes a 16K card they call "the Language Card" because it lets you use Pascal. However, essentially all of the 16K cards let you use Pascal. Other manufacturers include Microsoft, Saturn, Legend, ALS, Mountain Computer, Coex, ComX, Davong, Microtek, MPC, JDR, STB, Wesper and Prometheus.

Contrary to common belief, however, not all 16K cards are alike. Some do not have the D*/D switching feature. Yes, it's true that they're selling you 16K of RAM, but only 12K is accessible. Most software that use the high 16K of RAM assumes that it will be able to do D*/D switching and will crash horribly if you buy a Saturday Night Special 16K card. Few companies will admit to the manufacture of these "funky RAM cards," but there are hundreds of them in circulation.

Strapless RAM Cards

Most 16K cards have what is called a "strap" which you have to use to install the card. The way this works is that you remove one RAM chip from your motherboard and reinsert it in a special socket on the RAM card. A cable from the RAM card is then plugged into the empty chip socket on the motherboard (see Figure 25.3). The reason for this practice is that there are several signals available at the RAM sockets which are not available on the slot connector. The most important is the organization of the full 16 bit address into an eight bit column address and an eight bit row address and also the Row Address Strobe (RAS; see Chapter 21).

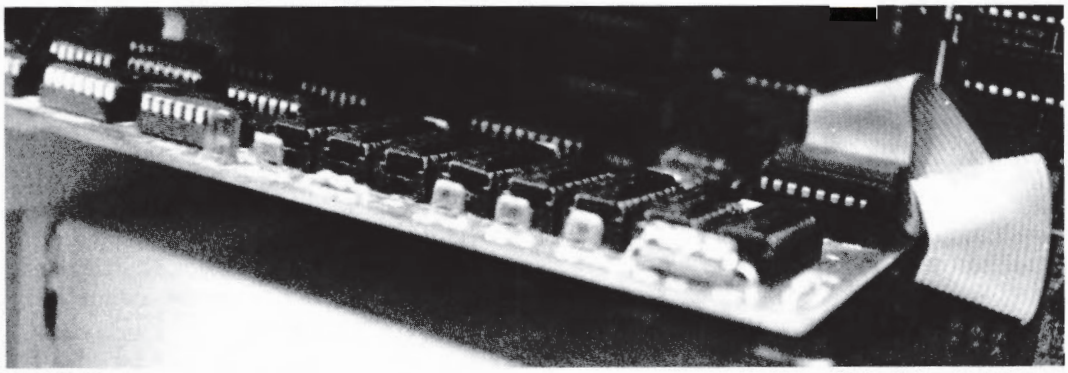


Fig. 25.3 The "strap" at the forward end of the 16K card carries several special RAM control signals out from the socket of one of the RAM chips. Strapless RAM cards generate these signals themselves, require no pulling of chips, and so, are easier to install.

It is possible to include on the card enough circuitry to regenerate the row and column addresses as well as the RAS signal using only information readily available on the peripheral connector. This makes it possible to manufacture a "strapless" RAM card which requires no pulling of chips or plugging in cables. The problem is that this does add to the cost of the card.

Few 16K cards are made in a strapless version, and there's really no reason to try to seek one out. The major interest in strapless RAM cards is for products which have much larger amounts of memory and which are intended to be compatible with the //e. It's not that there's any fundamental problem with using a "strap" in a //e (as long as the card handles 64K RAM chips properly), it's just that most of these cards would need a very long cable to reach the 64K RAM chips on the front of the //e.

High Capacity RAM Cards

There are a variety of memory expansion cards for the Apple II/II+ and //e which range from 32K up to one megabyte of RAM. Any owner of an Apple II/II+ who plans to get a high capacity card will probably get a card which uses some extended version of the D*/D-E-F bank switch system to select among multiple 16K banks. Apple //e owners can choose between one of these cards and a card which goes into the auxiliary slot and uses both an extension of this system and the auxiliary bank switch system discussed in Chapter 26.

Although many //e owners will want to have at least 64K of extra RAM in the auxiliary slot, there are some RAM card features not yet available on auxiliary slot cards, and there are other features which probably cannot be made available on the auxiliary slot unless it is redesigned by Apple.

The four high capacity RAM cards which provide the greatest versatility and which can serve as models for discussing the numerous other RAM cards are the Neptune Card and the Titan card from Titan Technologies (formerly Saturn Systems), the S'Card from Legend Industries, and the Know-Drive from Abacus Enterprises which companies are based in Ann Arbor, Pontiac, and Detroit, Michigan, respectively. This represents an unusual turn from the average Southern California state of affairs in these things, and it's worth repeating just for emphasis that Abacus is from Michigan, not Taiwan.

Of these four cards, the Neptune 192K card is for the //e auxiliary slot and so will be discussed in Chapter 26. The S'Card is interesting because it is the first Apple RAM card designed to accept either 64K RAM chips or 256K RAM chips. Although the price of 256K RAMs is still dropping as they go into mass production, only S'Card owners can buy now at 128 or 256K and upgrade later to one megabyte.

Powerful RAM Extension with the Know-Drive

The Know-Drive (see Figure 25.4) is the first really novel card in the Apple RAM market in many years, and there will probably soon be imitations from other companies. Along with 128K of RAM, the folks at Abacus have included the ability to operate an Apple signal called Non-Maskable Interrupt (NMI; see Chapter 27) which has the effect of abruptly stopping the Apple in the middle of whatever it is doing. The Know-Drive will then copy the complete contents of the Apple's main memory up into its own RAM chips and substitute a completely different 64K of information.

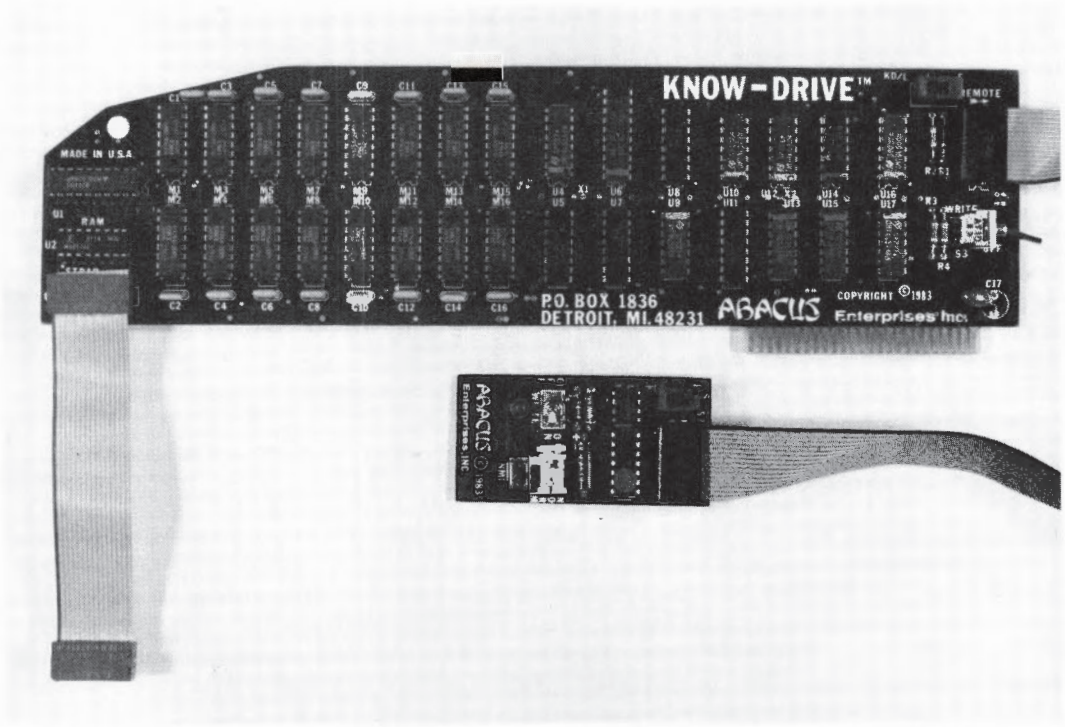


Fig. 25.4 Abacus 128K Know-Drive with external write protect and NMI switches. The strap can be used with Apple II/II+ or //e computers.

This means that you can be running a large VisiCalc model, hit a switch (in this case a real external switch that you hit with your finger), and pop into the middle of a game you were running earlier, and, with for instance Hard Hat Mack in midair, snap back instantly to your VisiCalc model. This sort of "background-foreground" operation has never been possible before on an Apple. The "Back-to-Back" system works with any software independent of copy protection, etc. and represents an enormous enhancement to the Apple's versatility.

This can't be done from a card in the auxiliary slot of a //e because Apple decided not to provide the NMI signal on the auxiliary connector, in part because it had been virtually unused for years. The only other use for the NMI signal has been for making backup copies of protected software with one of the copy cards (Crack-Shot, Back-It-Up, Wildcard, Alaska Card) or for stopping a program to print a copy of what is on the screen (Tex Print; see Chapter 18). The Know-Drive is compatible with all Legend and Saturn type memory extension software, can operate as a "RAMdisk" with DOS, ProDOS, CP/M and Pascal, can do the NMI back up trick, but is the first to offer background/foreground Apple operation. The basic Know-Drive lists for $485, and can be extended to 512K with an additional piggy-back board.

The Legend versus the Saturn Bank Select Systems

The bit oriented system for switching the D*/D-E-F banks of RAM has been extended to permit a program to switch really large amounts of RAM in and out of the highest 12K of the Apple's address space. Unfortunately, there are several different versions of this extension. The system developed by Saturn (Titan Technologies) for their various II/II+ RAM cards is the most widely used. This results in software compatibility among 128K cards from Saturn, Prometheus (Expand-a-RAM), and Omega (Ram-ex).

Unfortunately, this system was designed at a time when 128K was considered to be a reasonable maximum for a RAM card. The problem with 128K is that you can't quite copy an entire floppy disk onto it. All back-up via the RAMdisk must be done on a file by file basis.

Legend Industries anticipated the progressive improvements in price/storage in RAM chips and designed a system which can manage up to four megabytes. Don't laugh, there is very substantial interest in Legend's one megabyte S'Card and the other companies will be forced to redesign both their hardware and their software if they want to compete. The Legend system is a little bit more expensive, which is probably why the various other companies chose the Saturn approach.

The new Neptune card from Saturn for the //e has been redesigned to allow bank selection of up to 16 megabytes and they've already gotten underway on rewriting their software. Abacus has built their card to operate in either a "Saturn" or a "Legend" mode, so folks can buy software from either company or from third parties who have designed programs to work with these cards. However, Abacus has added a third system which lets them write software which will not run on any other RAM card.

Operating the Bank Select Lines

You will recall from Figure 25.2 that the bit position at A2 was ignored by 16K systems, and it was noted that it serves as a "bank select trigger." In the Saturn and in the Legend bank select schemes, a "one" in the A2 position changes the interpretation of bits A3, A1 and A0. The three schemes are laid out in Figure 25.2.

The Titan Card from Saturn System (see Figure 25.5) uses each of the eight unassigned bit patterns to select one of its 16K banks. This selection is made before actually configuring and executing the "bank switch" into the 6502's memory space.

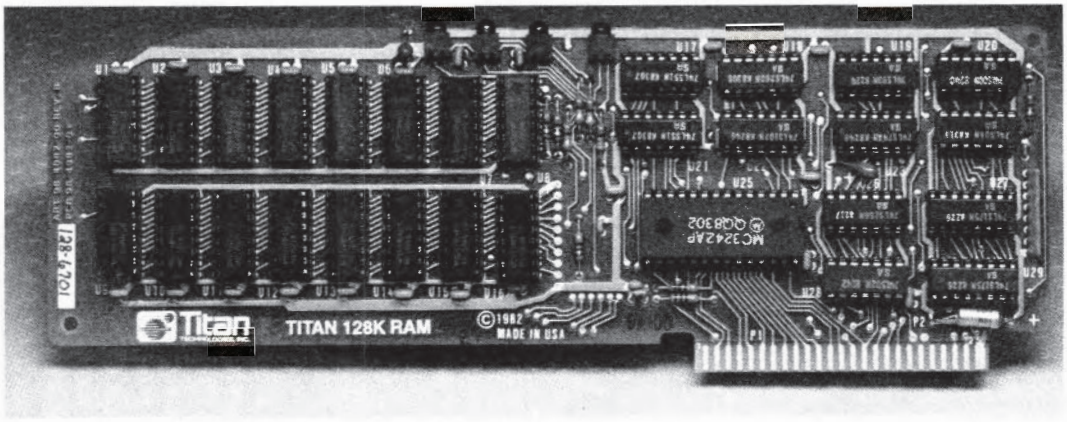
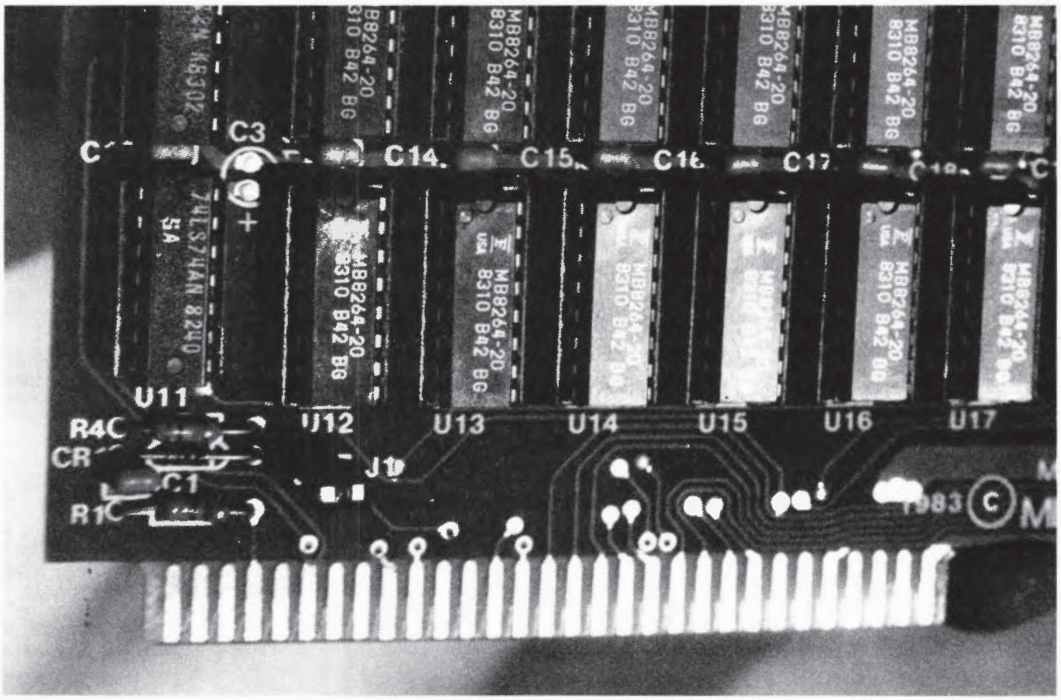


Fig. 25.5 Strapless 128K Titan Card from Saturn Systems.

Unlike the other systems, the Legend and Know-Drive approaches are not bit oriented. To select a 16K bank with a Legend card you address $C084, and the card then grabs a byte off of the data bus, thus making $C084 an effective "data port." The byte that gets latched in can have any value from 0 to 255, and each number points to a different 16K bank. Thus, the Saturn system provides for eight banks while the Legend system provides for 256 banks.

The Know-Drive system is similar to the Legend approach, in that it also latches a byte off the data bus. The difference is that any one of eight possible addresses with the A2 trigger set will cause it to grab a byte. This means that it will respond to Legend software which calls $C084, but it has permitted Abacus to write software which calls $C08F instead. Therefore, Legend software works on the Know-Drive, but Know- Drive software will not work on the Legend card. Further, an external switch can be used to reconfigure the Know-Drive so that it operates in the bit oriented Saturn mode instead to use Saturn-type software.


C3
C14
C15
C16
C17
U11
U12
U13
U14
U15
U16
U17
R4C
CR
C1
R1C
J1
MB8264-20
8310 B42 BG
74LS74AN 8240
1983 C

Chapter 26

On-Board Ports, Softswitches and Auxiliary RAM

The On-Board I/O Space

The heart of hearts of Apple control is in the lower part of the $C0 page. As you can see from Figure 26.1a, a lot goes on down there. Scattered among the locations are controls, inputs and outputs which affect one, the Game Port; two, the //c Mouse; three, the Speaker, Cassette and Utility ports; four, the //e and //c Auxiliary Memory Bank Switching; five, the Video Mode Controls; and six, the Keyboard and //c Serial Ports.

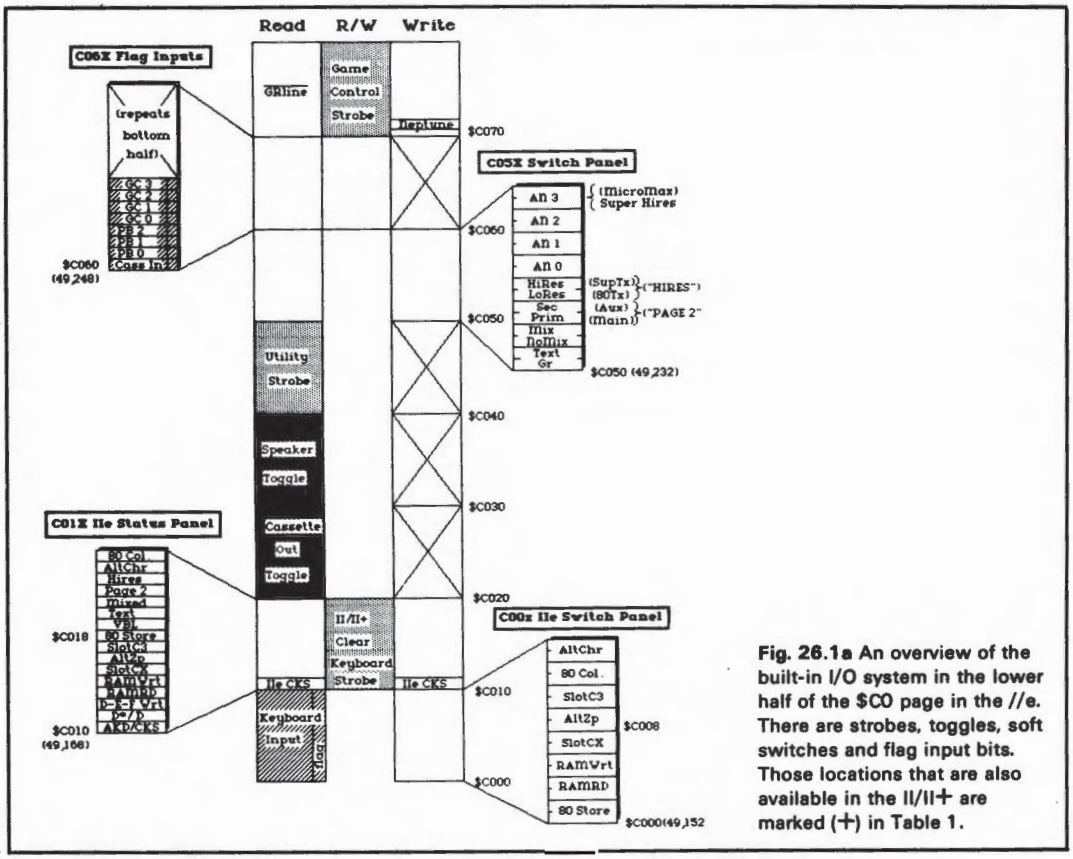


Fig. 26.1a An overview of the built-in I/O system in the lower half of the $C0 page in the //e. There are strobes, toggles, soft switches and flag input bits. Those locations that are also available in the II/II+ are marked (+) in Table 1.

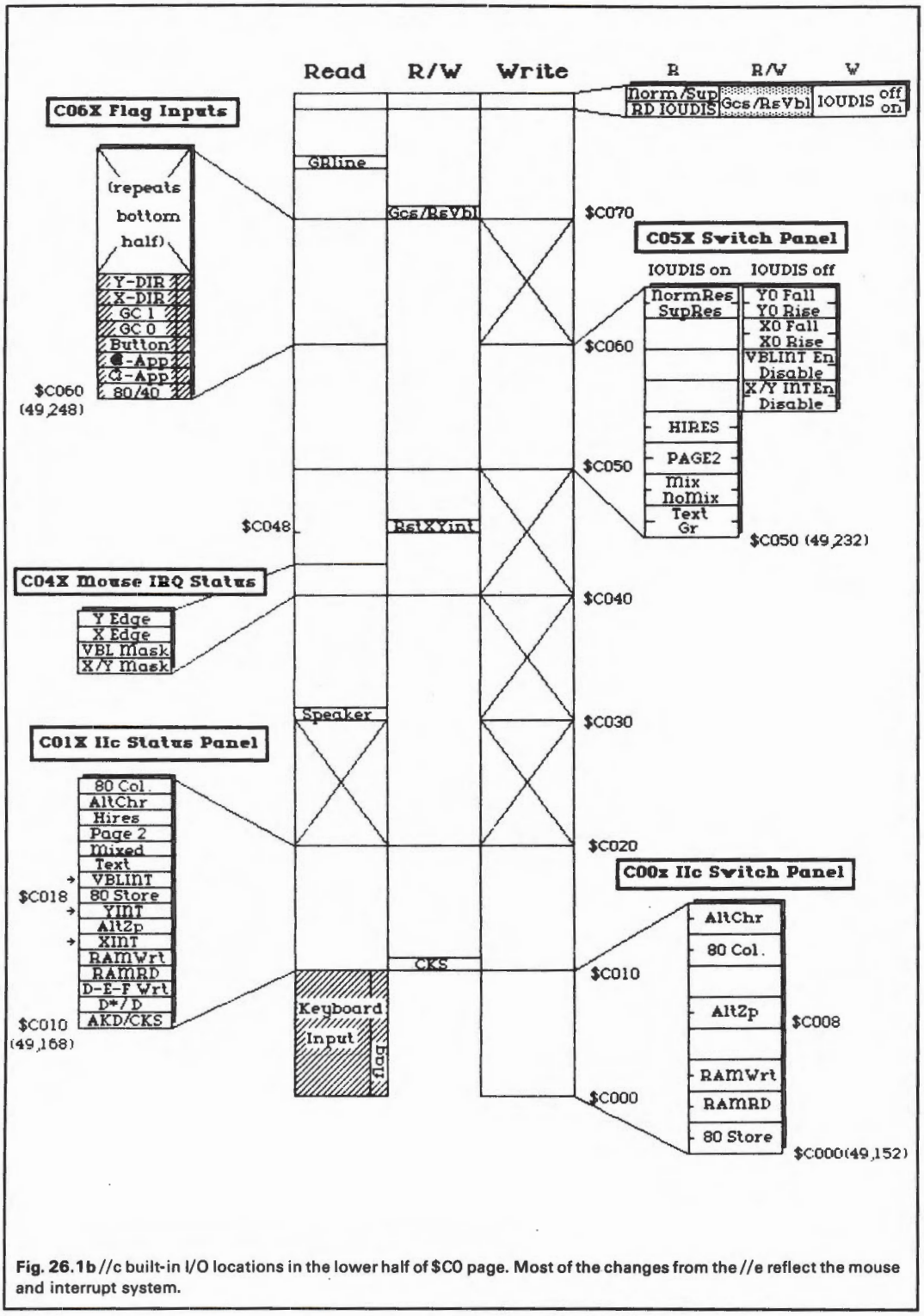


Fig. 26.1b //c built-in I/O locations in the lower half of $C0 page. Most of the changes from the //e reflect the mouse and interrupt system.

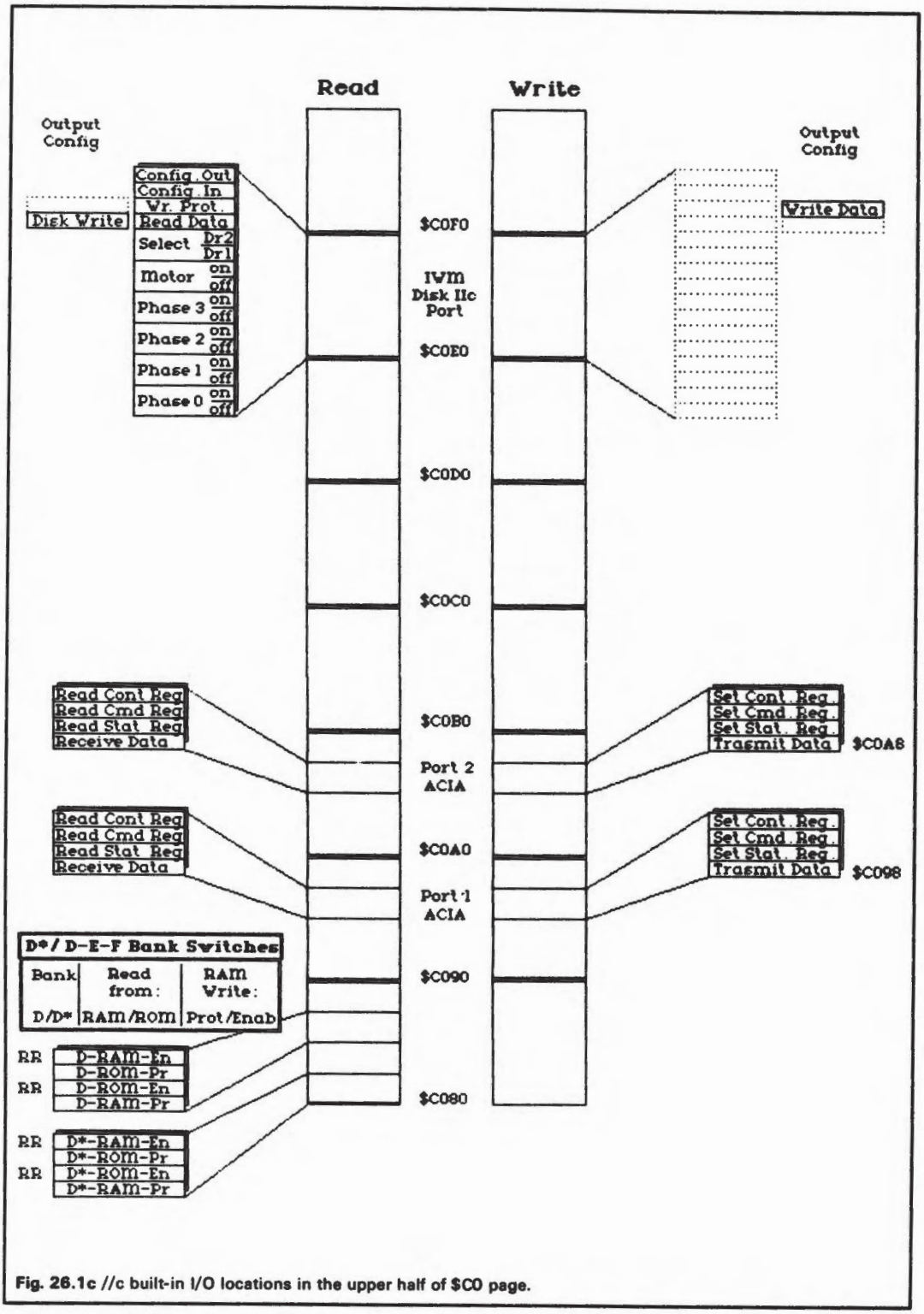


Fig. 26.1c //c built-in I/O locations in the upper half of $C0 page.

As in the upper half of the $C0 page, there is no RAM and no ROM in this space. The things which you actually find in these locations are strobe outputs, toggle outputs, softswitch flag outputs, flag inputs, and full byte input/output ports, which are discussed below.

A "strobe output" sends out a brief 0.98 microsecond pulse when it is addressed by a "read" (an LDA or a PEEK). Because of peculiarities of the 6502, the strobe will sometimes send out two pulses in very rapid succession if it is addressed with a "write" (an STA or a POKE). These are not true reads or writes since no data actually moves; rather, it is the act of addressing the location that is important.

"Toggle outputs" use a strobe to set their flip-flop. This means that each "read" to the address causes the output to flip from 0 to 1 or from 1 to 0, depending on how it happens to be set when the signal arrives. As with strobes, a toggle is affected by the double pulses that can occur during writes. Toggles differ from strobes in that they maintain their output state until addressed again. Figure 26.1a shows which strobes and toggles must only be "read" and which can be addressed either way.

Each "softswitch" output has two control addresses. One turns it "on" to a 1 output, while the second address turns it "off" to a 0 ouput. In the //e and //c these softswitches usually are also associated with a third address for "status," as opposed to control. The status address lets a program actually read the current state of the softswitch without causing its setting to change.

The softswitch "status" locations and several other "flag inputs" in the $C0 page all provide just one bit of data, the eighth bit. If the eighth bit is 0 then they are reporting an "off." If the eighth bit is 1 then they are reporting an "on." In BASIC, a PEEK statement will find X values between 0 and 127 if the input is reporting "off," and values between 128 and 255 if it is "on." Assembly language programs can use the BIT command followed by a BMI to branch if "on," or by a BPL to branch if "off."

Built-In I/O Ports and Control Locations

IIe (II/II+)			IIc
	$C07x Strobe and IIc Config. Panel		
	$C07F (W)	49,279	IOUDIS off (Mouse IRQ Config.)
	$C07E (W)	49,278	IOUDIS on (Access Norm/Sup)
	$C07F (R)	49,279	RD Norm/Sup
	$C07E (R)	49,278	RD IOUDIS
	$C077 (R)	49,271	RD GRline
RD GRline (inverted)	$C07x (R)	49,264->49,279	
(+)Game Control Strobe	$C07x (R/W)		GCS & Reset VBL IRQ
	$C06x Flag Input Panel		
(+)Game Control Input 3	$C067	49,255	Mouse Y-DIR
(+)Game Control Input 2	$C066	49,254	Mouse X-DIR
(+)Game Control Input 1	$C065	49,253	same
(+)Game Control Input 0	$C064	49,252	same
(+)Push Button Input 2	$C063	49,251	Mouse Button
(+)Push Button Input 1	$C062	49,250	Solid Apple Key
(+)Push Button Input 0	$C061	49,249	Open Apple Key
(+)Cassette Input	$C060	49,248	80/40 Switch

Table 26.1 Built-in I/O and control locations.

IIe (II/II+)				IIc	
$C05x Switch Panel					
		$C05F (W)	49,247	Y0 IRQ edge = fall	(IOUDIS off)
		$C05E (W)	49,246	Y0 IRQ edge = rise	(IOUDIS off)
		$C05D (W)	49,245	X0 IRQ edge = fall	(IOUDIS off)
		$C05C (W)	49,244	X0 IRQ edge = rise	(IOUDIS off)
		$C05B (W)	49,243	VBL IRQ Enabled	(IOUDIS off)
		$C05A (W)	49,242	VBL IRQ Disabled	(IOUDIS off)
		$C059 (W)	49,241	X0/Y0 IRQ Enabled	(IOUDIS off)
		$C058 (W)	42,240	X0/Y0 IRQ Disabled	(IOUDIS off)
(+)Annunciator 3	On	$C05F (W)	49,247	Normal Graphics	(IOUDIS on)
(+)Annunciator 3	Off	$C05E (W)	49,246	Super Res Graphics	(IOUDIS on)
(+)Annunciator 2	On	$C05D (W)	49,245	absent	(IOUDIS on)
(+)Annunciator 2	Off	$C05C (W)	49,244	absent	(IOUDIS on)
(+)Annunciator 1	On	$C05B (W)	49,243	absent	(IOUDIS on)
(+)Annunciator 1	Off	$C05A (W)	49,242	absent	(IOUDIS on)
(+)Annunciator 0	On	$C059 (W)	49,241	absent	(IOUDIS on)
(+)Annunciator 0	Off	$C058 (W)	49,240	absent	(IOUDIS on)
(+)Hires (SupTx)		$C057 (W)	49,239	same	
(+)Lores (80Tx)		$C056 (W)	49,238	same	
(+)Secondary(Aux)		$C055 (W)	49,237	same	
(+)Primary (Main)		$C054 (W)	49,236	same	
(+)Mix		$C053 (W)	49,235	same	
(+)NoMix		$C052 (W)	49,234	same	
(+)Text		$C051 (W)	49,233	same	
(+)Gr		$C050 (W)	49,232	same	

Table 26.1 (continued).

IIe (II/II+)			IIc
$C04x Strobe & IIc IRQ Panel			
	$C048 (R/W)	49,224	Reset X0/Y0 IRQ
	$C043 (R)	49,219	RD Y0 IRQ edge (1 = fall)
	$C042 (R)	49,218	RD X0 IRQ edge (1 = fall)
	$C041 (R)	49,217	RD VBL IRQ Enable/Disable
	$C040 (R)	49,216	RD X0/Y0 IRQ Enable/Disable
(+)Utility Strobe	$C04x (R)	49,216 ->49,231	absent
$C03x Toggle			
(+)Speaker Output Toggle	$C03x (R)	49,200 ->49,215	same (use $C030)
$C02x Toggle			
(+)Cassette Output Toggle	$C020 (R)	49,184 ->49,199	absent
$C01x IIe Status Panel			
RD 80 Col.	$C01F (R)	49,183	same
RD AltChr	$C01E (R)	49,182	same
RD HiRes/LoRes (HIRES)	$C01D (R)	49,181	same
RD Sec/Prim (PAGE2)	$C01C (R)	49,180	same
RD Mix/NoMix	$C01B (R)	49,179	same
RD Text/Gr	$C01A (R)	49,178	same
RD VBL	$C019 (R)	49,177	RD and Reset VBL IRQ
RD 80 Store	$C018 (R)	49,176	same
RD SlotC3	$C017 (R)	49,175	RD Y0 IRQ
RD AltZp	$C016 (R)	49,174	same
RD SlotCX	$C015 (R)	49,173	RD X0 IRQ
RD RAMWrt	$C014 (R)	49,172	same
RD RAMRD	$C013 (R)	49,171	same
RD D-E-F RAM Prot/En	$C012 (R)	49,170	same
RD D^{II}/D	$C011 (R)	49,169	same
RD AKD	$C010 (R)	49,168	same
(+)Clear Keyb. Strobe	$C010 (R/W)	49,168	same

Table 26.1 (continued).

IIe (II/II+)				IIc
$C00x IIe Switch Panel				
AltChr	On	$C00F (W)	49,167	same
AltChr	Off	$C00E (W)	49,166	same
80 Col.	On	$C00D (W)	49,165	same
80 Col.	Off	$C00C (W)	49,164	same
SlotC3	On	$C00B (W)	49,163	absent
SlotC3	Off	$C00A (W)	49,162	absent
AltZp	On	$C009 (W)	49,161	same
AltZp	Off	$C008 (W)	49,160	same
SlotCX	On	$C007 (W)	49,159	absent
SlotCX	Off	$C006 (W)	49,158	absent
RAMWrt	On	$C005 (W)	49,157	same
RAMWrt	Off	$C004 (W)	49,156	same
RAMRD	On	$C003 (W)	49,155	same
RAMRD	Off	$C002 (W)	49,154	same
80 Store	On	$C001 (W)	49,153	same
80 Store	Off	$C000 (W)	49,152	same
$C000 Input				
(+)Keyboard Data and Flag		$C000 (R)	49,152	same

Table 26.1 (continued).

The Game Port Paddle Analog to Digital Converters

At the top of Figure 26.1a, the addresses the $C07x panel are identified as the "game control strobe," and the four addresses in the $C06x panel labeled GC0, GC1, GC2 and GC3 are the the "game control analog inputs." (These are also referred to as "paddle" inputs PDL1, PDL2, etc.). Note in Figure 26.1b that the //c has got only GC0 and GC1.

This is system that monitors joysticks and trackballs and the like, but which can also be used to measure a very wide variety of continuously varying electrical or mechanical events in the outside world, all depending on what device is attached to the game port. The operation of this system is explained in some detail in Chapters 9 and 14.

Do be sure to note, however, that although the mouse plugs into the same external port on the //c, the analog to digital converter system is deactivated whenever the mouse is attached. The mouse uses a much more precise positioning system described below and in Chapter 9.

In the Apple's "analog to digital converter" system, an external device converts some quantity, such as the deflection of a joystick, into a "resistance" in a circuit which runs back into the Apple's game port. Different resistances signify different angles of the joystick. The Apple's converter measures the resistance.

To take a measurement, you start by addressing a location called the "Game Controller Strobe." As you can see from Figure 26.1a, in the II/II+ and //e it doesn't matter exactly what address you send as long as it's between $C070 and $C07F (49,264 and 49,279). In the //c, however, it is recommended that only $C070 be used (see Figure 26.1b).

This strobe signal clears the converter for a new measurement and also causes a "one" to be put into the eighth bit of each of the game control inputs (GC0 to GC3). It is the timing of the change of these "input flag" bits back to zero which will serve as the source of information for the program.

The converter responds to a resistance between 0 and 150K ohms by assigning to that resistance a proportional time value between 0 and 3.062 milliseconds. At the end of the appropriate amount of time it changes the value of the eighth bit at the game control inputs. All four of the game control inputs at $C064 through $C067 (49,252 through 49,255) are measured simultaneously.

A program tests one of the analog ports by sending a $C07x game control strobe and then doing a program loop which repeatedly checks the appropriate game control inputs to see if they have changed yet. The number of passes through your program's loop tells you the position of the joystick. The PDL 0 function in Applesoft BASIC will do all this for you automatically, and assembly language programmers can use the PREAD routine at $FB1E.

There are two other important notes here for the //e. The game control strobe output is also available on one of the pins of the auxiliary slot connector. Therefore, it is used for special purposes by auxiliary slot cards such as the Neptune memory card and will be discussed shortly. The other point is that the $C07x addresses can be used to set up an "undocumented //e video mode" described later in this chapter.

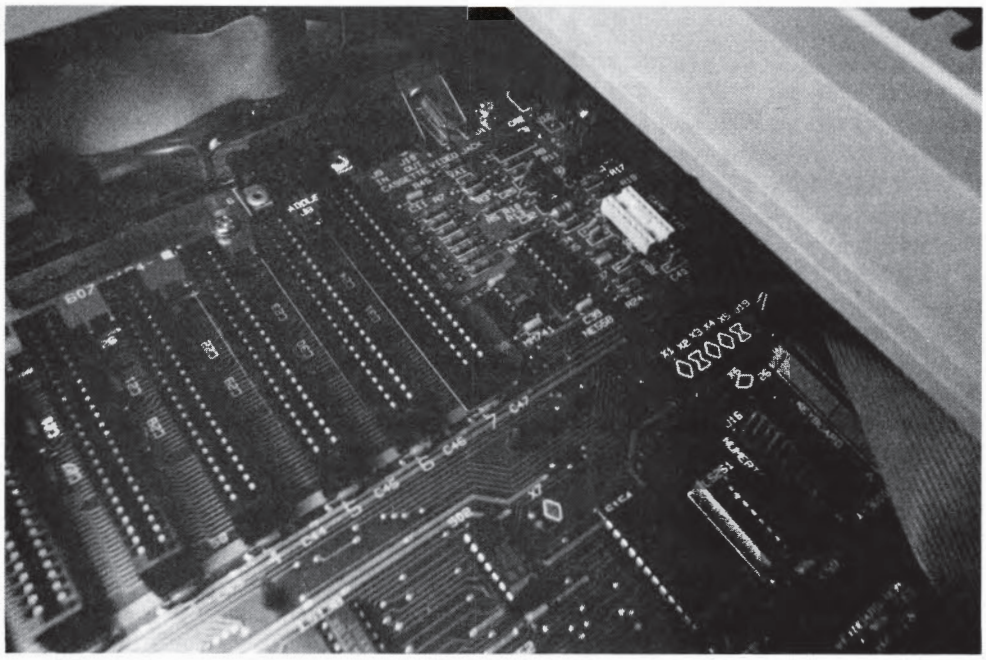


Fig. 26.2a //e internal game port connector and I/O circuitry.

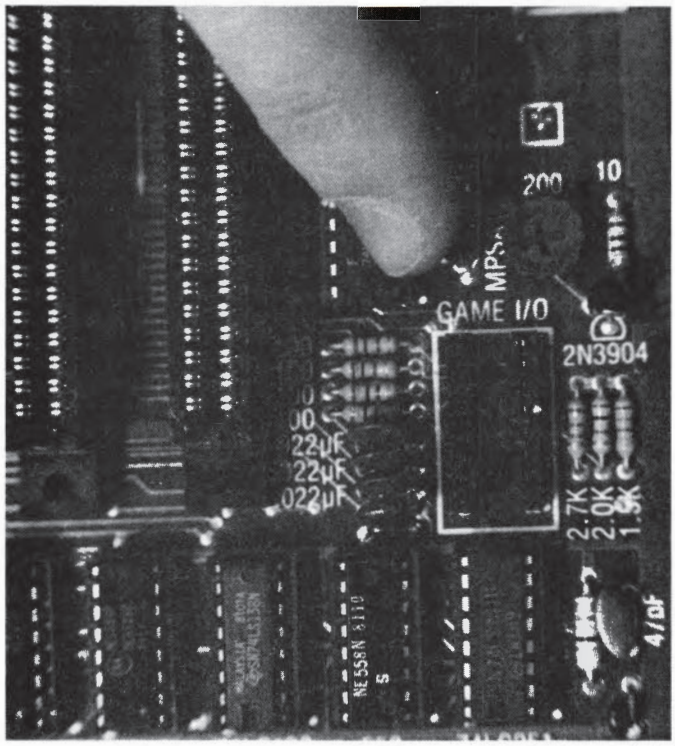


Fig. 26.2b II/II+ game connector. Resistors, capacitors, and 558 timer make up the A/D converter for the game paddles.

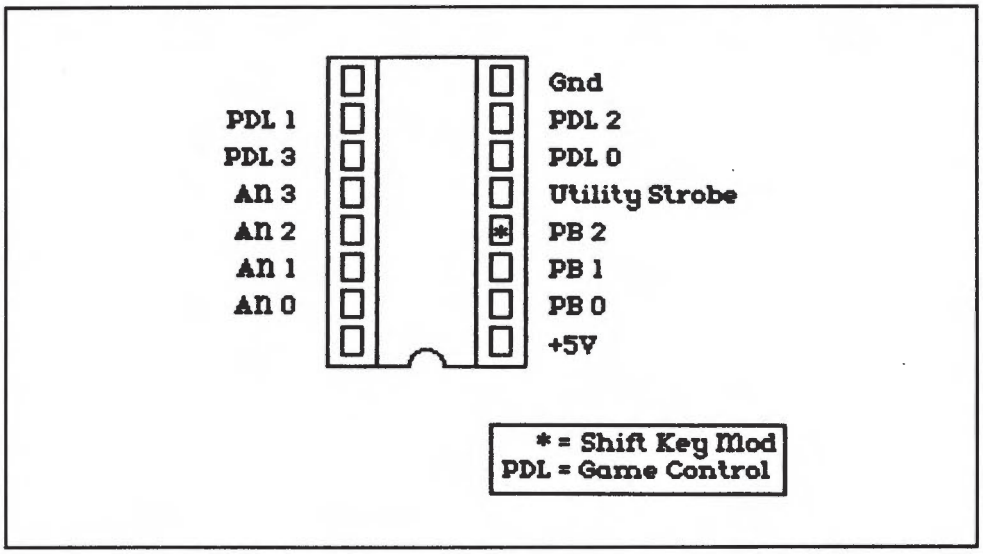


Fig. 26.2c Game connector pinout. The notch is on the keyboard end of the connector. If you use stranded wire for the shift key, there's room for the wire and a pin from a game controller in the PB2 (push button 2) position.

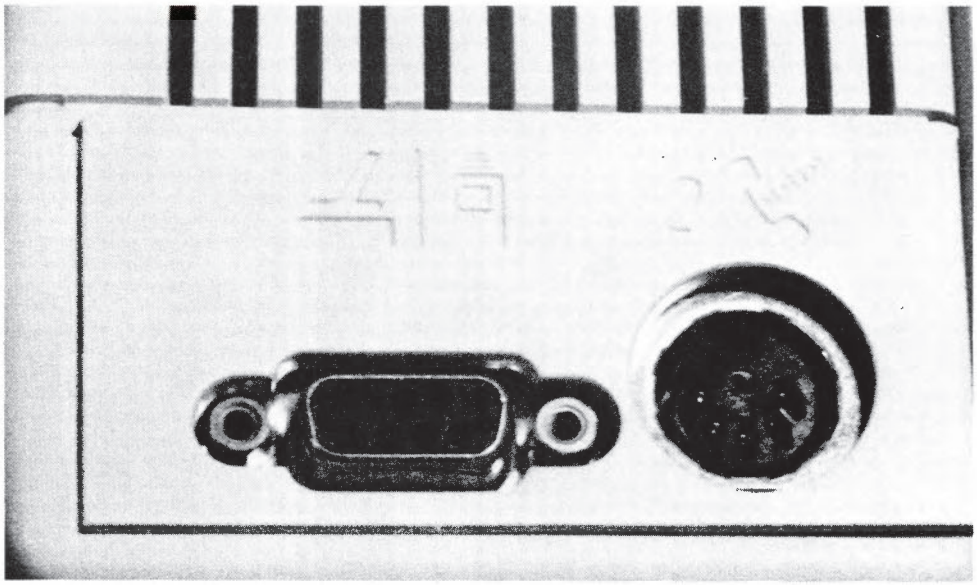


Fig. 26.2d The //c mouse and game paddle port. Although the two devices share the same external plug-in port, the electronics, speed, accuracy, and internal systems are completely different. When the mouse is connected, an optically coupled switch is thrown, and effectively rewires the internal connections.

Pushbutton Flag Inputs

Just below the game control inputs in Figure 26.1a you can see three locations labeled PB2, PB1 and PB0; these are the "push button inputs" ($C061 to $C063, 49,249 to 49,251). They are much simpler than the game control inputs. If someone out there is holding down a button on a game paddle, then there will be a "one" at the eighth bit of the appropriate push button input. With the finger off the button, a quick read reveals a "zero" in the eighth bit of that location.

In the Apple II/II+, PB0 and PB1 are usually used for game control buttons, but PB2 (see Figure 26.4c) is used as the input for the "shift key mod." You run a wire from the shift key to PB2 (see Chapter 8, Figure 26.4 and see Appendix D), and a program can check the state of PB2 after each keypress to see if you want upper or lower case. In the //e and //c, PB0 is connected to the "Open-Apple" key and PB1 is connected to the "Closed-Apple" key.

When the mouse is connected in the //c, PB2 ($C063) turns on (eighth bit "1") and stays on except when you're pressing the mouse button. In addition, the //c has a fourth push button input at location $C060 (49,248), a location that has been taken over from the old II/II+, and //e cassette tape input. The $C060 input in the //c is operated by the 80/40 switch on the keyboard, and this is the only thing in the Apple that is affected by pushing that switch. If you push it down, you set the eighth bit at $C060, and programs can interpret this as your way of saying that you've only got a low resolution TV out there, so please don't try to send out 80 column text.

The Annunciator Softswitch Outputs

The softswitches in an Apple each provides a means for the 6502 to send out a steady TTL standard output of "zero" (0 volts) or "one" (5 volts) to some device which is not within its address space. Most of softswitches have permanent and important assignments for switching functions on the Apple motherboard, but the four softswitches between $C058 and $C05F (49,240 and 49,247) are left unassigned in an Apple II/II+. These four switches are labeled as AN0, AN1, AN2, and AN3 for "annunciators" 0, 1, 2 and 3 in Figure 26.1a. Their outputs are available on the internal game connector for any digital TTL device which needs direct control signals from the 6502.

The //e does have an assigned use for one of these, AN3. In a properly configured //e (see Appendix D) it is used to control super high res graphics ($C05E equals Super, $C05F equals standard). The //c also uses the AN3 locations for a super high res switch, but it has a more complex set of rules of operation—which are reviewed in the section below on the //c Mouse. AN0, AN1 and AN2 are removed completely from the //c.

Softswitch Hardware

The softswitches in an Apple II/II+ are made from a chip called an "addressable latch" (74LS259). The chip has eight different output latches, but just one data input line. A one or a zero can be fed in from the input line to be stored in whichever one of the output latches has been selected. That stored value will be continually announced by the selected latch's output pin.

The latches are selected by three address lines. In the Apple II/II+, address signals A3, A2 and A1 from the main address bus are used to select which of the eight outputs will collect data from the chip's single input. Notice that A0 is not used in the select operation. This

means that the latch cannot tell the difference between, say, 49,244 and 49,245. This is why both of those addresses are called Annunciator 2 and indeed all of the softswitches take up two locations.

The input on the chip is connected to address line A0. This means that when the latch is selected with an even number (such as 49,244) the data input will receive a 0, (all even binary numbers end in 0). The odd number just after that (49,245) will select the same part of the chip, but in this case the result will be to place a "one" at that output.

In the //e and //c these softswitches have disappeared into the interior of the Memory Management Unit (MMU), the Input/Output Unit (IOU), and the General Logic Unit (GLU), but they still operate in the same fashion. The only difference is that the eight switches in the $C00x panel (i.e., with addresses between $C000 and $C00F; locations 49,152 and 49,167 in decimal) must always be addressed with a "write" operation. Reads or writes will work just fine for the eight switches in the $C05x panel.

Hardware Controls for the //c Mouse Port

The mouse may be wonderfully easy to use as a pointing device, but it is marvelously complex at the level of hardware operation. For this reason, Apple has provided a substantial amount of built-in firmware in the //c which programmers can use without any detailed knowledge of the hardware involved. These firmware routines and their relation to the hardware are explored in considerable detail in Chapter 9.

There are five data inputs from the mouse. One is an up/down signal for the button which is read like a pushbutton input at $C063 (49,251). It is unusual only in that it is inverted from the usual sense of an Apple pushbutton; normally eighth bit high, but dropping to zero when the button is down.

The remaining four inputs concern mouse movement and mouse direction. As the mouse moves, various wheels and axles spin inside the mouse housing to produce two pairs of signals: X-MOVE and Y-MOVE; X-DIR and Y-DIR. (These lines have abbreviated names as follows: X-MOVE = X0, Y-MOVE = Y0, X-DIR = X1, Y-DIR = Y1).

The X-MOVE and Y-MOVE lines send pulses to the Apple as the mouse moves along right/left, and forward/backward axes (see Chapter 9). The X-DIR and Y-DIR lines do exactly the same thing except that, for instance, X-DIR turns on a few instants before X-MOVE when the motion is to the left, but turns on a few instants after X-MOVE when the motion is to the right. Y-DIR works with Y-MOVE in a similar way.

The X-DIR and Y-DIR signals are read as if they were simple pushbutton inputs at $C066 and $C067, respectively. The problem is that they must be read within 40 microseconds (millionths of a second) of any change from the other pair of inputs (X-MOVE and Y-MOVE). Because of the precise time requirement, the AppleMouse interface card for the Apple //e has its own 6805 microprocessor. In the //c, however, the 65C02 does all the work and an elegant interrupt system is used to assure that X-DIR and Y-DIR are read quickly before their information becomes invalid.

The X-MOVE and Y-MOVE signals come directly into an interrupt generation section of the //c IOU. There, each is associated with three softswitches. The first softswitch determines whether the IOU will respond to the "rising" or the "falling" edge of the incoming pulse signsl. The locations that control this function are at $C05F/$C05E for the Y-MOVE (Y0) line and $C05D/$C05C for the X-MOVE(X0) line (see Figure 26.1b and Table 1B).

Once the selected edge has arrived from X0 or Y0, the IOU must be told whether to ignore the input or to respond by generating an interrupt with its IRQ line to the 65C02. This choice is determined by the setting of a single softswitch for both the X0 and Y0 lines. A write to $C059 enables X0/Y0 interrupts, while a write to $C058 disables these move interrupts. When they are disabled, no meaningful data can be collected from the mouse so this pair can be thought of as an on/off switch for the mouse.

When a mouse movement causes an interrupt, the 65C02 stops what it is doing (quickly grabs data from X-DIR and Y-DIR just in case) and starts scanning through the system to check all possible sources of interrupts. The X-MOVE and Y-MOVE inputs each have an "interrupt source flag" location that they turn on when they succeed in generating an interrupt. The 65C02 looks at these flags in the course of its scan, and if it finds a 1 in the eighth bit of $C015 (X0) or $C017 (Y0), it has found the source of the interrupt. (Note: these are documented incorrectly in some versions of the //c technical literature. They are read locations.)

Once the 65C02 has located the source of the interrupt it goes about updating its record of mouse position. However, before it finishes and returns to the program that was running before the interrupt, it must turn off the IRQ signal coming from the IOUs. To do this, it does a read or write to $C048 which "resets" the X0 and Y0 interrupt lines.

VBL Interrupt Locations for //c Mouse Routines

Since X0/Y0 interrupts must be enabled for the mouse to function, a mouse program in the //c would get new data at unpredictable and occasionally quite frequent intervals. However, a running program really shouldn't update anything on the video screen more than 60 times a second, and it should do the updating during the "vertical blanking period" when the CRT's electron gun is off (see Chapters 5 and 9). The Apple interrupt handling firmware is therefore designed to operate in a mode in which it can "service" all X0 or Y0 interrupts "transparently," without making the running program aware of what is going on, unless a vertical blanking period is occurring.

Thus, the Vertical Blanking (VBL) signal is made part of the mouse input system. This 60 Hz signal is generated inside the IOU, but it comes with a set of softswitches much like those for X0 and Y0. You must first enable or disable VBL interrupts ($C05B/$C05A). The 65C02 can check for VBL interrupts at the "interrupt source flag" at $C019, and it can reset them by an address to $C070.

One crucial difference between the VBL and the X0/Y0 interrupt locations is that when you read the VBL source flag at $C019, you also reset it, just as you do with $C070. This is the cause of a real problem for software compatibility between the //e and //c. The //e permits a read of VBL at $C019, but the signal always stays on for the entire duration of VBL. In the //c, you don't know when the vertical blanking period ends because $C019 only turns on at the rising edge of the system VBL signal, and it is reset (cleared) as soon as it is read.

Getting Access to Mouse $C05x Locations

All of the control softswitches for the IOU interrupts are in the top half of the $C05x panel (see Figure 26.1b). These four pairs of locations handle Y0 edge fall/rise ($C05F/$C05E), X0 edge fall/rise ($C05D/$C05C), VBL interrupt enable/disable ($C05B/$C05A), and X0 or Y0 interrupt enable/disable ($C059/$C058).

The catch is that all four of these softswitches are inaccessible unless you first set a completely different switch called IOUDIS, which is at $C07F/$C07E. When IOUDIS is on, you disable access to these four interrupt control switches in the IOU. In this state, the $C05F/$C05E locations at the top of the $C05x panel act like the AN3 switch in a //e (see Figure 26.1a and 26.1b). As in the //e, this switch selects super high res graphics ($C05E) or standard high res graphics ($C05F).

As a consequence, you should always use a special sequence of instructions when you want to change the setting of one of the mouse interrupt switches. These instructions are as follows:

```
STA $C07E     ;Enable IOU access
STA $C05x     ;Set your chosen interrupt switch
STA $C07F     ;Disable IOU access
```

GAME PORT I/O and Control

Inputs

Clear Game Cont. Strobe	$C070 (->F)	49,264 (->79)
Game Control Input 3	$C067	49,255
Game Control Input 2	$C066	49,254
Game Control Input 1	$C065	49,253
Game Control Input 0	$C064	49,252
Push Button Input 2	$C063	49,251
Push Button Input 1	$C062	49,250
Push Button Input 0	$C061	49,249

Outputs

Annunciator 3 On	$C05F	49,247
Annunciator 3 Off	$C05E	49,246
Annunciator 2 On	$C05D	49,245
Annunciator 2 Off	$C05C	49,244
Annunciator 1 On	$C05B	49,243
Annunciator 1 Off	$C05A	49,242
Annunciator 0 On	$C059	49,241
Annunciator 0 Off	$C058	49,240
Utility Strobe	$C040 (->F)	49,216 (->31)

Speaker Port

Speaker Output Toggle	$C030 (->F)	49,200 (->15)

Cassette Tape Port

Cassette Input	$C060	49,248
Cassette Output Toggle	$C020 (->F)	49,184 (->90)

Keyboard Port

AKD/Clear Keyb. Strobe	$C010	49,168
Keyboard Data and Flag	$C000	49,152 (read only)

Table 26.2 Built-in port controls.

The Utility Strobe, Speaker Toggle, and Cassette Port

Skipping over the remainder of the softswitches in the $C05x panel for now, we come to the "Utility Strobe" at $CO4x (49,216 to 49,231) and the "Speaker Toggle" at $C03x (49,200 to 49,215) (see Figure 26.1a). The utility strobe is another nice, solid TTL output that the 6502 can control. It is different from the annunciators in that it is used for brief pulses rather than for setting a steady level. Normally, the utility strobe line on the game connector puts out a steady +5 volts. When it is addressed with a read instruction, it drops to 0 volts during "phase 0" (see Chapter 5), which lasts for about half a microsecond. The Utility Strobe has been removed in the //c and the $C04x addresses have been reassigned to various mouse functions described elsewhere in this chapter.

The speaker location, which includes any address from $C030 to $C03F, is a "toggle" flip-flop. When it is addressed with a read, its output changes. If it was 0, the act of addressing it changes the output to 1. If it was 1, then it gets changed to 0. This output is connected to the speaker circuitry in such a way that each flip produces a click from the speaker. You can control the frequency of the sound coming from the speaker by addressing the speaker from a machine language loop. The larger and slower the loop, the lower the tone. Operation of the speaker is identical in the //c, except that programmers are supposed to use only the $C030 address (see Figure 26.1b).

The Apple II/II+ and //e Monitor provides machine language routines for saving and retrieving data with a standard cassette tape recorder. The $C060 location is labelled in Figure 26.1a as the cassette flag input, and it changes back and forth between 1 and 0 as the data bits stream in from the tape. Data is stored by flipping the toggle addressed at any $C02X location.

Back in the days before disk drives, this was it. A few folks still keep a cassette recorder near their Apple to save critical information in case of a disk disaster, but these ports don't get too much use these days. This whole system has been removed from the //c. $C060 has been reassigned as a pushbutton input for the 80/40 switch on the keyboard, and the $C02x locations are reserved. The //c IOU has an output that responds to $C02x, but it is not connected to anything.

Video and Memory Control with Softswitches

The remaining softswitches can be considered in three categories: Program Memory Bank Switches, Display Memory Bank Switches, and Video Output Mode Switches. In the //c, all of these memory bank switches are for controlling various parts of the Auxiliary RAM which is made up of the eight RAM chips towards the back end of the machine (see Chapter 1, Figure 1.1d and 4B). In the //e, they control RAM on the "text card" or "64K extended text card" in you auxiliary slot, but there are also switches for some ROM in the $C000 space (see Chapter 22, Figure 22.1a).

Some of the Program Memory Bank Switches, and all of the Display Memory Bank Switches, act to let the 6502 read or write from Auxiliary RAM. The only thing special about the Display Memory switches is that they act only on those areas of RAM that are used to store text or graphics data intended for use by the video generation system (see Chapter 5). The Display Memory ranges in Auxiliary RAM are marked Hi Res Screen 1X and Text Screen 1X in Figure 26.3.

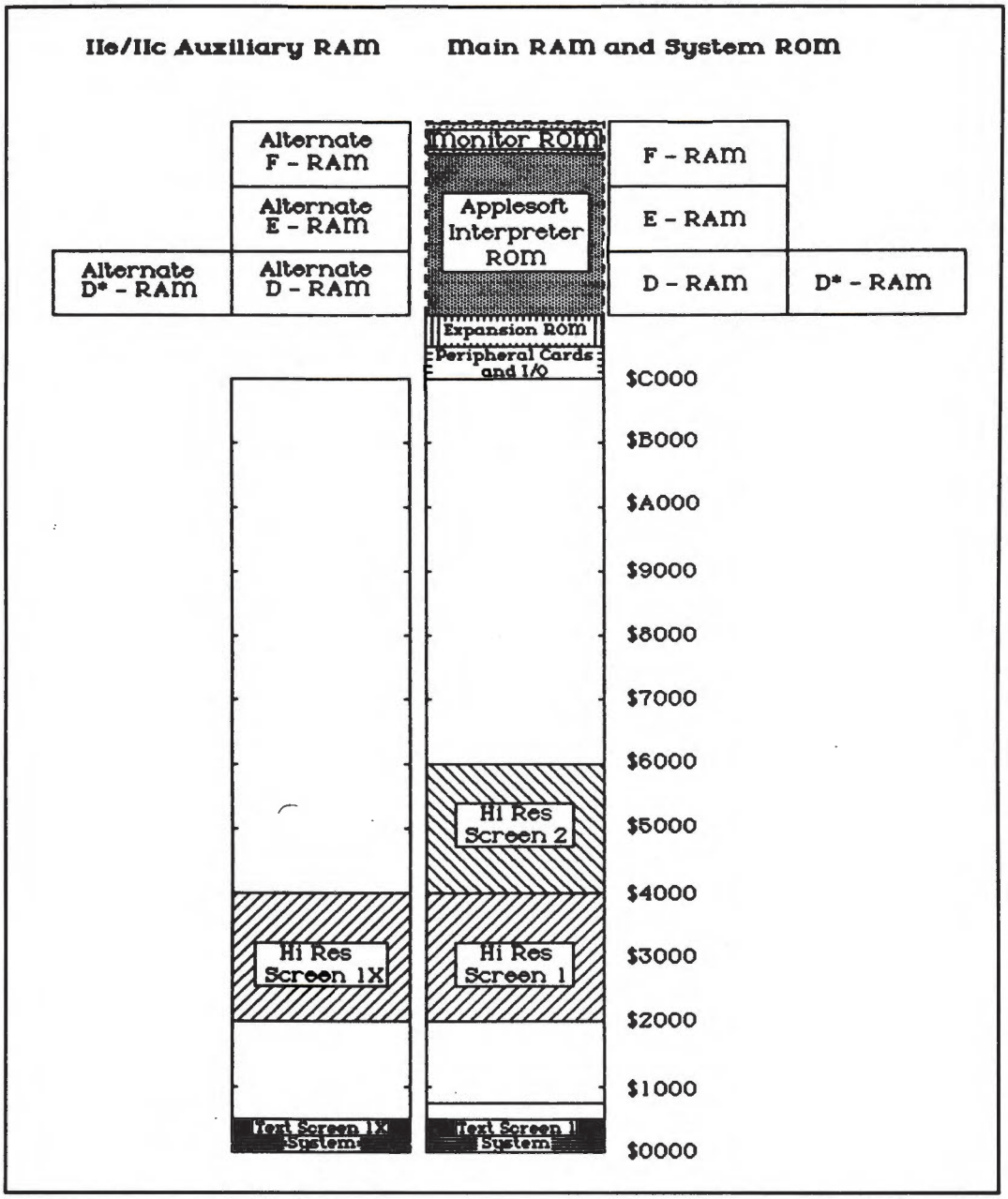


Fig. 26.3 Full memory map of 128K Apple //e and //c. Sixteen K of ROM and I/O locations plus 128K of RAM are bank switched in and out of the 6502s single address space.

Hardware Details of Auxiliary RAM Bank Switching

Many of the fundamentals of bank switching in general are covered in Chapter 25. Bank switching is a means of using a single 64K address space as a sort of window that can be moved around over a much larger amount of RAM or ROM. In //e and //c Auxiliary RAM bank

switching, all addresses to RAM simultaneously select one location in Main RAM and an alter ego location in Auxiliary RAM which has the same address. However, only one of the two is able to actually exchange data with the 6502 at any given instant.

There are two parts to this control of data flow, one is important only for "Writes" by the 6502, and one is important for both reads and writes. The Auxiliary RAM chips are connected to an "alternate data bus" (see Appendix A). Whenever the 6502 attempts to read from RAM, the Auxiliary chips dump a byte of data onto this alternate data bus. When the read is actually intended for chips in Main memory; however, the bank switching system keeps the alternate data bus disconnected from the 6502's main data bus. Thus the auxiliary chips announce their byte but there is no one listening.

The Alternate Data Bus

The alternate data bus and the main data bus connect to two opposite sides of a 74LS 245 chip. The bank switching system acts on this chip to control whether or not data can pass between the 6502 and the alternate data bus. The actual control is carried out by two signals, "EN80" and "R/W80" (see Appendix A). The EN80 signal comes from the MMU and it is responsibe for connecting the alternate data bus to the main data bus.

The 74LS 245 connects all eight data lines simultaneously, but it has two different ways of making the connection. One takes data from the left side of the chip and amplifies it and sends it off to the right (so to speak), while the other takes data from the right, amplifies it, and sends it off to the left. In the //e and //c, the two "sides" represent a write where data moves from the 6502 to the Auxiliary RAM or a read where data moves from the Auxiliary RAM to the 6502.

The R/W80 signal starts out initially as the R/W (read/write) signal from the 6502 (see Chapters 21 and 27); however, it is "gated" by the EN80 signal. When EN80 is off and Auxiliary RAM is not being used, then R/W80 never makes it to the 74LS 245. When EN80 is on, it lets the 6502's R/W signal pass to the 74LS 245 as the "R/W80" signal whereupon it configures that chip for the appropriate direction of data flow.

That is how the connection is managed for bank switching, but there are two more considerations. First, the Main RAM and the Auxiliary RAM must not be able to get data onto the main data bus at the same time. Second, the Auxiliary RAM must not attempt to read data when there is nothing on the alternate data bus for it to look at.

Disabling Main RAM and Blocking R/W

The first problem is solved by disabling the Main RAM whenever EN80 is turned on. The MMU sends a signal called CASSEN over to the PAL (TMG in the //c) and this causes the Column Address Strobe (CAS) for the Main RAM to shut down. As outlined in Chapters 21 and 25, CAS must be sent to a RAM chip at a very precise moment or that chip will never be able to tell if it is being addressed. The CAS for the Auxiliary RAM is never turned off.

In actual fact, the CAS input to chips in Auxiliary RAM comes from another Apple timing signal called "Q3." What is "Q3?" Well, it's really just an old timing signal that was lying around unused from early Apple II days. Designers of the Auxiliary RAM system spotted its potential and put it to work in their bank switching scheme.

The other potential problem (keeping the Auxiliary RAM chips from grabbing data off of a disconnected alternate data bus) is solved by the EN80 and R/W80 lines. The problem could arise if the chips thought there was a "write" from the 6502 going on and thus that they were supposed to grab up some new data. To keep this from happening, they are usually told that the 6502 is doing a read and that they shouldn't grab any new data. Only if EN80 turns on R/W80, and the 6502 has set its R/W line to W, will the Auxiliary chips grab up data.

The Video Data Bus

One final note worth including in this section is that all of these details about bank switching apply only to the 6502's use of the RAM. When the Video Display system is using the RAM, things are quite different.

Characters for odd columns in the display are stored in Main RAM while characters for even columns are stored in Auxiliary RAM. As with the 6502 system, the Video Display Generator simultaneously selects the Main and Auxiliary RAM location with the same address. When this happens, the chips in Main RAM dump their byte onto the main data bus and the chips in auxiliary RAM dump their byte onto the alternate data bus.

At that moment, the video system activates two different 74LS 374 chips, one for the main data bus and one for the alternate data bus. When the event is over, one LS 374 chip holds a byte for an even column while the other holds a byte for the following odd column. The video system then uses the video data bus attached to the other side of both LS 374s to ship first the even and then the odd data to the video signal generation section (see Chapter 6 and Appendix A).

Program Memory Bank Switches

The Program Memory Bank Switches come in three sets. One set is concerned with managing the high 16K of Auxiliary RAM, which must be coordinated with the D*/D-E-F bank switching described in Chapter 25. This high 16K of Auxiliary RAM is sometimes referred to as "Alternate RAM." The second set is used in the //e only and it acts on the internal ROMs for $C000 space. The third set controls most of the lower 48K of the Auxiliary RAM.

The //e and //c Alternate RAM Select System

The Alternate RAM Select system is an extension of the D*/D-E-F Bank switching arrangement described in detail in Chapter 25. In part it merely integrates the hardware details of Auxiliary RAM switching with the older switching system addresses that dates back into the dawning days of the Apple II. However, it also provides an important means of helping to make sure that the Monitor and Applesoft Interpreter have not been accidently disconnected.

The switch that controls the Alternate Bank is called Alternate/Zero Page (ALTZP) and is located at $C008/$C009 (49,160/49,161). It determines whether the D*/D-E-F switch settings will apply to the high 16K on the motherboard Main RAM chips or to Auxiliary RAM. When the ALTZP switch is turned off, the D*/D-E-F switch settings apply to the motherboard RAM. This is completely independent of the other switches which control the rest of the Auxiliary RAM.

In addition to the alternate D*/D-E-F space control, the ALTZP switch controls what RAM will be in the $00 page and $01 page (see Figure 26.4). These two pages of address space have special significance for the 6502 (see Chapters 21 and 27). The locations within these 512 bytes are used for a variety of very critical "scratch pad" functions. Keeping the switch off helps protect the programmer from disaster.

Any critical operations done by the 6502 or the system software while these pages are occupied by auxiliary RAM (ALTZP on) will require carefully copying the changed values into the two pages in the main RAM to avoid confusion later. By keeping the zero page and stack page ($01) separate from the switching of the remainder of the free RAM area, the //e and //c auxiliary RAM has been made into a much safer environment for programmers.

It is also possible to use the ALTZP system in very sophisticated system level work in which

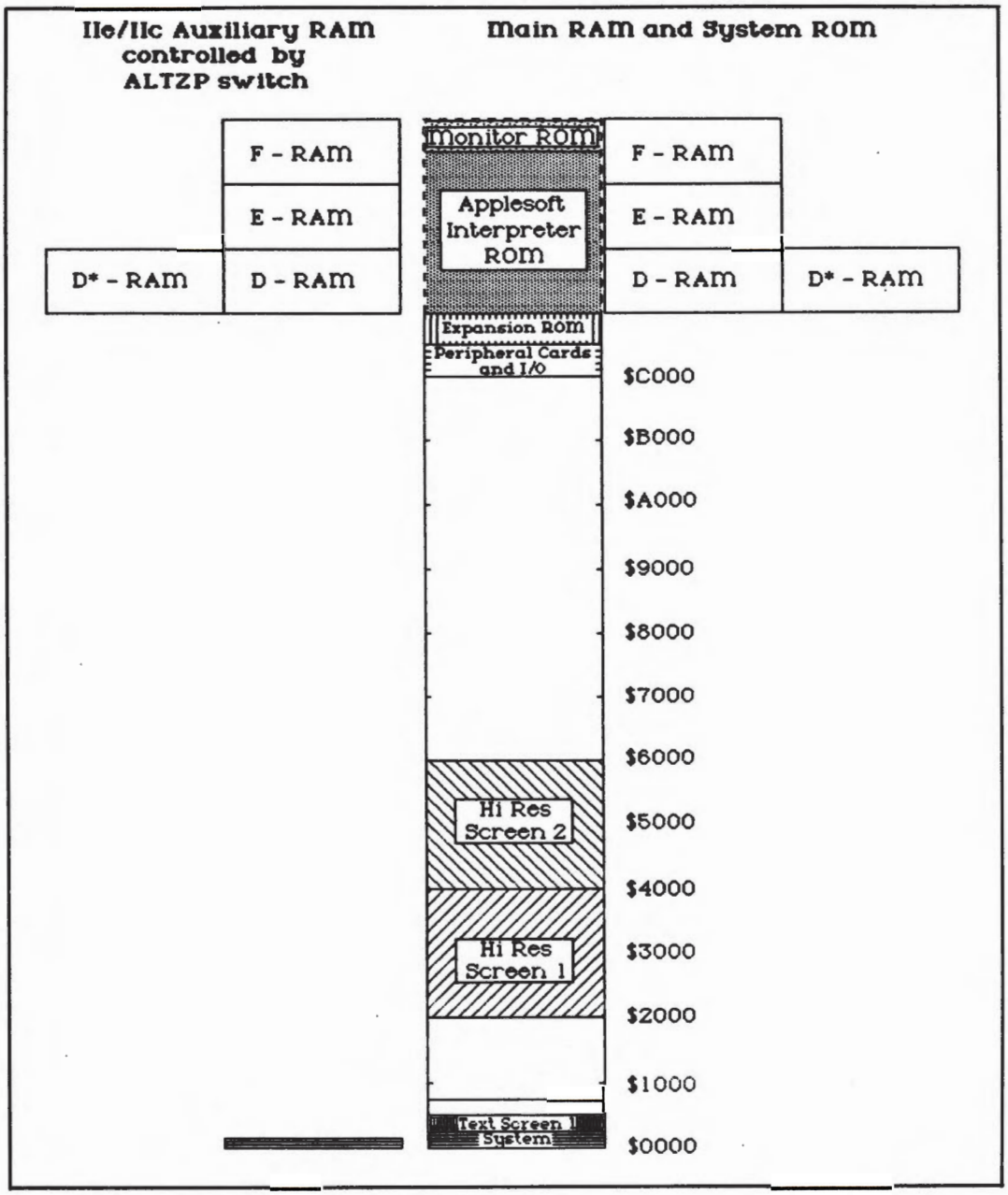


Fig. 26.4 The //e and //c RAM controlled by ALTZP bank switches.

two different operating environments, each with its own zero page, stack, and high 16K of RAM, are running at the same time, doing very different things.

As an additional fringe benefit in the //e, the ALTZP system makes it possible to confidently use a D*/D-E-F RAM card (such as the Legend S'Card) in one of the standard slots while using RAM in the auxiliary slot for display purposes.

Bank Switching the //e Internal $C000 ROMs

The internal $C000 ROM switches in the //e attend to two special problems in //e operation. The first is that much of the Apple and other commercial software must treat the //e 80 column text card as if it were in slot 3 instead of in the auxiliary slot.

When a slot is called from software, the Apple //e's motherboard decodes the address and sends an I/O Select signal to the slot to activate the card's 256 byte Peripheral Card ROM in the $CN00 space. This ROM can then get control of the 2K Expansion ROM space for the use of its expansion ROM and also go about operating its switches and ports in the Peripheral Card I/O ($C0n0) space (see Chapter 22 and Appendix B).

The first problem was how to ensure that the 80 column card in the auxiliary slot got turned on instead of some other card in slot 3. The //e tests for the presence of a card in the auxiliary slot, and if there is one there it prepares to redirect the I/O Select signal so that it will activate the auxiliary card's $C300 ROM rather than the ROM of a card in slot 3. The redirection of the I/0 Select line is controlled by a switch called "SLOTC3ROM." This switch is located at 49,162/49,163 ($C00A/$C00B).

The folks at Videx, who built an 80 column card for slot 3 in the //e, defeat this function by taking advantage of one of those little arcania of Apple function. It takes the Apple about 0.2 millionths of a second to determine that a $C300 address has been selected and then to throw the "SLOTC3ROM" switch. The Videx Ultraterm also watches for the address and is able to take action before the SLOTC3ROM switch is thrown. This makes it possible to use the Videx Ultraterm for high resolution text characters but still use the auxiliary RAM for super high res graphics. The contention between the auxiliary slot and slot 3 is over activation of the ROM, not over use of the auxiliary slot itself.

The other problem that Apple had to deal with in the //e is that they wanted to put a variety of things into ROM, but there just wasn't any more space. The solution was to install a complete set of replacement ROMs for the nearly 4K of ROM addresses in the $C000 space (see Chapter 22, Figure 22.1a). To use this replacement ROM, the //e uses its "SLOTCXROM" switch ($C006/$C007). This switch turns the I/O Select system off altogether. All of the seven Peripheral Card ROMs are thus disabled and the //e is able to take the entire space from $C100 to $C7FF for itself as well as seizing the Expansion ROM space. The nearly 4K of ROM is used in part for a few extra flourishes in the control program for the video display, but it also contains the self test programs that respond to the CTRL/Solid Apple/Reset sequence.

These switches disappear from the //c where there is only one set of permanent installed ROM in the $C000 space and no switching is ever done (see Chapter 22, Figure 26.1b). The switch locations in the $C00x panel are left unused, and the associated status locations in the $C01x panel are given over to the mouse system.

RAMRD and RAMWRT Switching System

The remainder of the Auxiliary RAM includes the "47 and a half K" between $0200 (512) and $C000 (49,152; see Figure 26.5). The two Program Memory Bank switches that control this expanse of memory are called "RAMRD" ($C002/$C003) and "RAMWRT" ($C004/$C005). When RAMRD is turned on, then any address in the auxiliary range applies to the auxiliary RAM if it's in a "read" instruction, but to motherboard RAM if it's with a "write" instruction. If RAMRD is off but RAMWRT is on, then the reverse is true. If both RAMRD and RAMWRT are on, then all reads and writes in this range will apply to the auxiliary memory only.

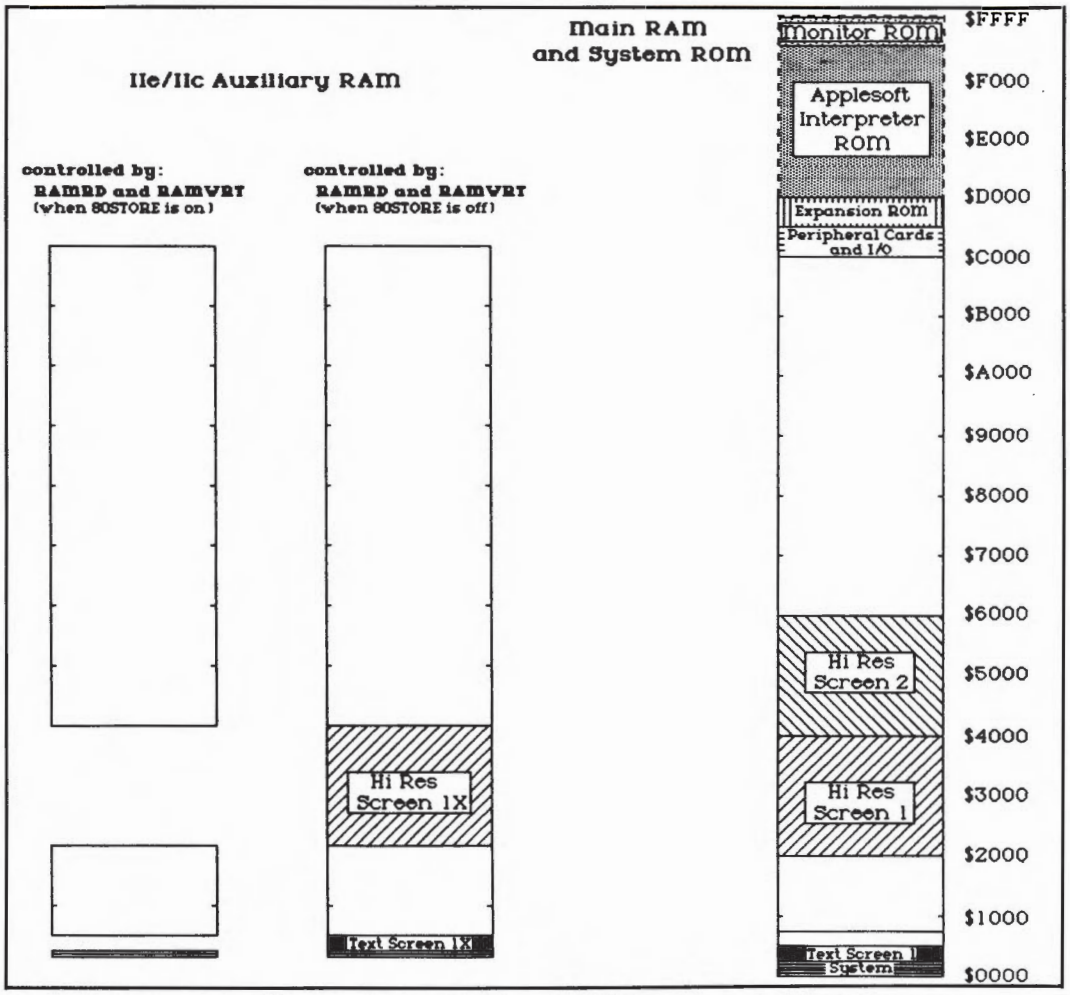


Fig. 26.5 The //e and //c RAM controlled by RAMRD and RAMWRT switches. The 80STORE switch can override RAMRD and RAMWRT so that the text and graphics software have independent control of the screen memory bank switching. The small patch of RAM at the bottom left is page $02 (input buffer) and $03 (routines and vectors).

//e and //c Display Memory Bank Switches

That's moderately straightforward, but as you can see from Figure 26.5, there is a potential problem. This range of memory includes all of the text and graphics display areas. Whenever the Apple is operating in 80 column or super high res mode, the 6502 must be able to store text or graphics data in the display areas in the auxiliary RAM. It is possible to do this with RAMWRT, but it is very easy for a programmer to get in big trouble fast by switching away access to the whole program space just to put some characters on the screen.

In order to help make the auxiliary display pages really usable by a large number of BASIC and assembly language programmers, Apple included a special subset of bank switches that affect only the display page areas. A great many BASIC programmers are already quite accustomed to keeping their programs and variables out of the main memory display areas. You can use the special display area switches with no more caution than you normally reserve for avoiding the display areas in main memory.

Using 80STORE

Having a special set of switches just for display memory was a great idea, but the problem was that there were not enough places left in the two switch panels ($C05x panel and $C00x panel). To make the system work, you need to have one switch that goes back and forth between main display memory and auxiliary display memory (see Figure 26.5), as well as a second switch that can select between enabling just the text screen area, or, for more advanced work, letting both the text screen and hi res screen one throw back and forth (see Figure 26.6). The only place left however, was a single switch at $C000/$C001, which has been labeled "80STORE" in Figure 26.1

To solve the problem, this 80STORE switch was set up to reconfigure the soft switch system rather than to do any bank switching itself. When 80STORE is turned on by a write to $C001 (POKE 49153,0) it changes the function of two switches in the $C05x panel. The two switches that get altered are at $C054/$C055 and at $C056/$C057.

The names for these two switches vary among Apple documents. In normal mode, with 80STORE off, the first of these ($C054/$C055) is usually called "Primary/Secondary" because it selects whether the video generator will scan hi res Display Screen one or hi res Display Screen two (see Figure 26.6) in all Apple IIs. The other one is usually called "Hi Res" or "Lo-Res GR/Hi-Res GR" because it selects between low resolution and high resolution graphics displays in all Apple IIs.

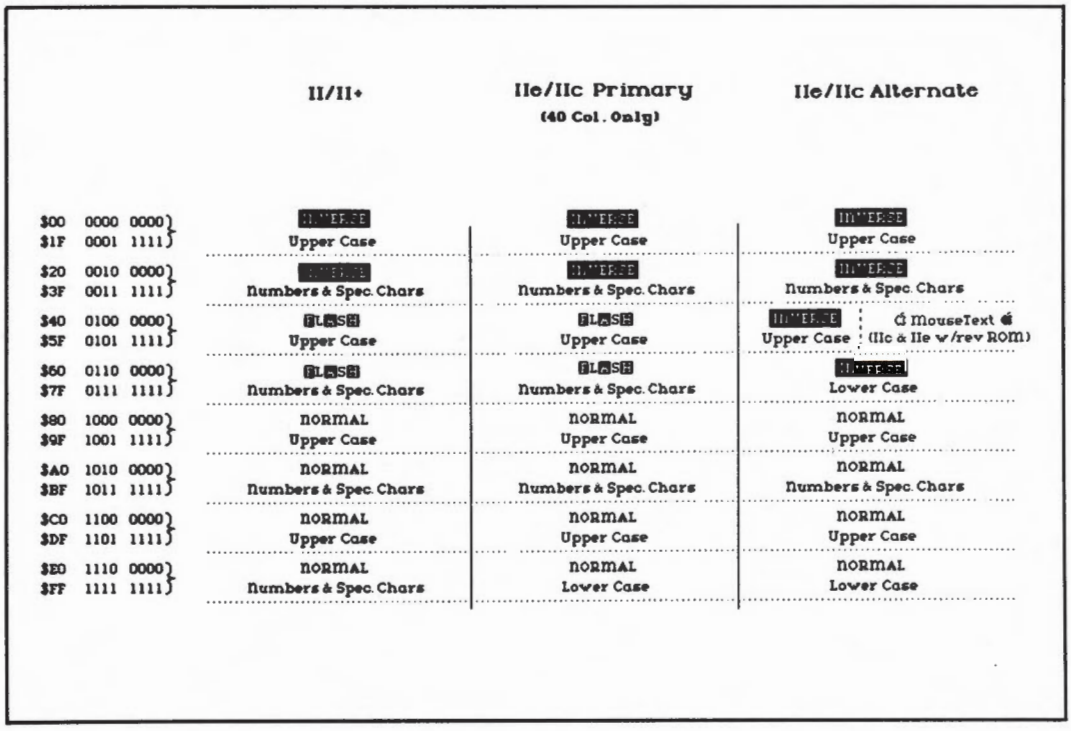


Fig. 26.6 The text and graphics routines use the PAGE2 switch (Sec./Prim.) as a bank switch to load data for even columns into auxiliary (1X) display memory, and data for odd columns into main memory. In this mode, it has no effect on scanning by the video system in the main memory, but has no control over auxiliary bank switching. With 80STORE on, the HIRES switch controls both bank switching during writes, and scanning by the video system.

With 80STORE turned on, the function of these two switches changes. The first one ($C054/$C055) could now be called "Main/Aux" because it actually does the bank switching between Auxiliary RAM and Main RAM. This switch is called "PAGE2" in the Apple //e and //c Reference Manuals. Note carefully that when 80STORE is off, this switch affects the only the scanning pattern of the video generator, but when 80STORE is on, it becomes only an Auxiliary RAM bank switch for the 6502, and has no affect on the video generator.

IIe/IIc Auxiliary Program Memory Bank Switching

SlotC3	On	$C00B	49,163	(IIe only)
SlotC3	Off	$C00A	49,162	"
SlotC3	Status	$C017	49,175	"
SlotCX	On	$C007	49,159	(IIe only)
SlotCX	Off	$C006	49,158	"
SlotCX	Status	$C015	49,173	"
AltZp	On	$C009	49,161	
AltZp	Off	$C008	49,160	
AltZp	Status	$C016	49,174	
RAMWrt	On	$C005	49,157	
RAMWrt	Off	$C004	49,156	
RAMWrt	Status	$C014	49,172	
RAMRD	On	$C003	49,155	
RAMRD	Off	$C002	49,154	
RAMRD	Status	$C013	49,171	

D*/D-E-F Bank Switches for II/II+/IIe/IIc

D-RAM-En	$C08B	49,291	(II/II+ with 16K card in slot #0)
D-ROM-Pr	$C08A	49,290	
D-ROM-En	$C089	49,289	
D-RAM-Pr	$C088	49,287	
D*-RAM-En	$C083	49,283	
D*-ROM-Pr	$C082	49,282	
D*-ROM-En	$C081	49,281	
D*-RAM-Pr	$C080	49,280	
Protect/Enable Status	$C012	49,170	(IIe/IIc only)
D/D* Bank Select Status	$C011	49,169	"

Table 26.3 //e and //c bank switching locations.

The other switch that is affected by 80STORE, $C056/$C057 could be called "80Tx/SupTx" in its altered state. When it is turned off with a write to $C056 (POKE 49238,0) then the Main/Aux switch controls only the text screen 1 display area. When it is turned on with a write to $C057 (POKE 49239,0) then the Main/Aux switch affects both text screen one and hi res screen one (see Figure 26.6).

When 80STORE is off, this switch selects whether the video generator will scan the lo res or the high res display memory areas. With 80STORE on, it affects both the scanning pattern of the video generator and also it afects how much memory PAGE2 will be able to bank switch. If you turn on 80STORE, activate graphics mode (see below), and set this switch to its 80Tx position, you will get "super lo res graphics" (see Figure 26.9)—one of the "undocumented video modes" discussed later in the chapter.

High Capacity Auxiliary RAM Cards and Disk Emulators

Plug-in RAM cards for the //e auxiliary slot are fairly simple in design, because nearly all of the work in the switching scheme described thus far is done by the MMU (see above). "Extended" 64K text cards are available from Apple, Coex, ComX, MPC and Quadram. Cards which provide extra functions along with the 64K are made by TSK and Apple, which includes an "RGB" color video system (see Chapter 7), and by Microsoft which includes a complete Z-80B CP/M system along with the 64K (see Chapter 30). However, there is no reason why you have to settle for just 64K out there in the auxiliary slot.

The tradition with high capacity RAM cards in the Apple II/II+ has been to let the user operate them either as an extension to the Apple's Memory or as a high speed disk drive emulator. ProDOS automatically turns the extra 64K of Auxiliary RAM into a simulated extra disk drive, but it is a shame to be limited to a 64K "drive" when the Apple floppies can hold 128K. As explained in Chapter 24, a RAM disk in the auxiliary slot is up to 10 times slower than the RAM disk speed champion, the PCPI RAM extender; nonetheless, it can be an enormous improvement over a floppy.

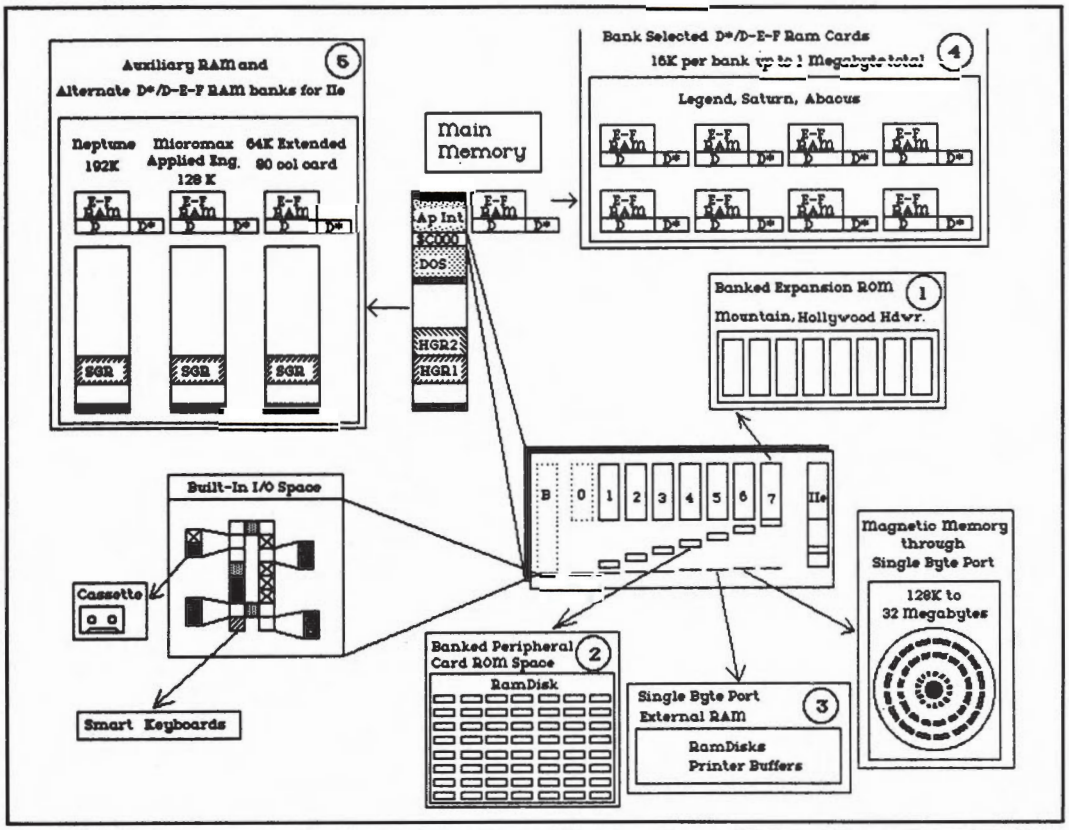


Fig. 26.7 Master memory expansion diagram. 1. (see Chapter 22), 2. (see Chapter 22), 3. (see Chapter 24), 4. (see Chapter 25), 5. (see Chapter 26).

Figure 26.7 above reviews some of the numerous ways Apple memory can be expanded. Of all of these, Auxiliary RAM cards offer the largest linear segment which can be switched at one time—48K with RAMRD and RAMWRT.

Cards for the auxiliary slot which have 128K of RAM are made by MicroMax (ViewMax 80-e, see Figure 26.8) and by Applied Engineering (Memory Master //e). Titan Technologies (formerly Saturn Systems) makes a 192K RAM card for the auxiliary slot called the Neptune card.

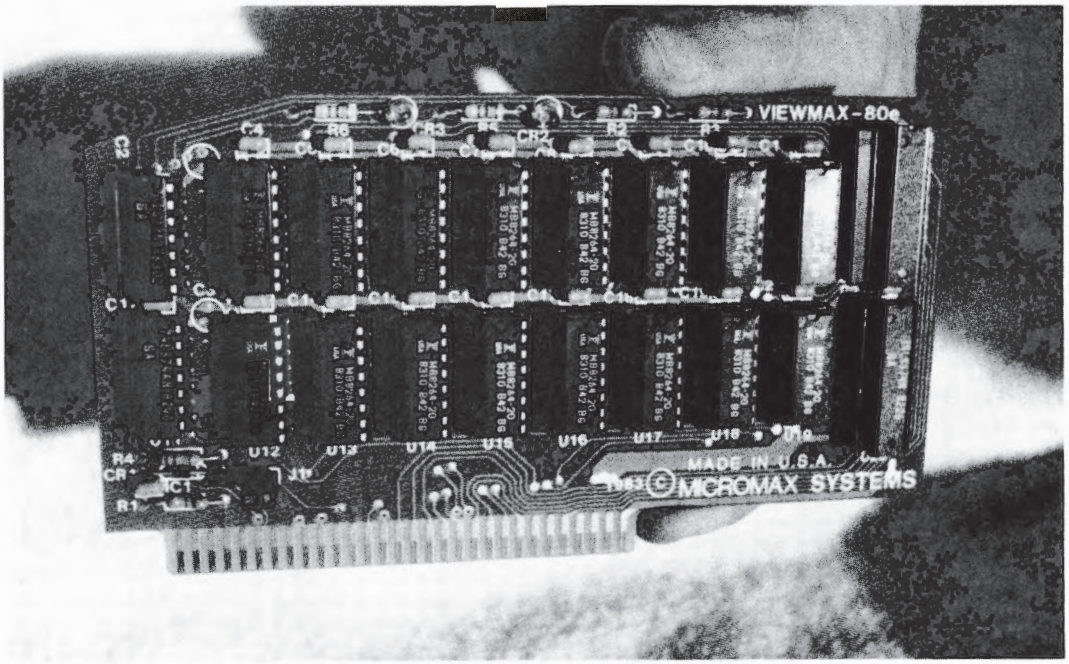


Fig. 26.8 Viewmax 80e from MicroMax provides 128K for the auxiliary slot. The small black jumper on the bottom left of the card enables super hi-res on Revision B //e's.

As many users of 128K RAM disk emulators have learned, you can't copy an entire Apple floppy into the RAM on a 128K card. The Neptune's extra space makes the card enormously convenient for backup. You can copy an entire disk out to the Neptune card, put in a blank diskette in your Disk II drive, format it, and copy the contents onto it directly from the Neptune card. The Neptune RAMdisk system is set up so that when you hit CTRL/Open Apple/Reset, nothing happens to the contents of the Neptune drive. It's just like leaving a disk in the drive while you clear main memory and start over.

Controlling Extra RAM in the Auxiliary Slot

Those of you who read through the section on bank switching in D*/D-E-F type RAM disks (see Chapter 25) will recall that there were a variety of different signaling schemes used which render software written for one card useless on the other types of cards (except the Know-Drive which can handle all the systems). Well, manufacturers of //e RAM disks are off to a fast start toward mutual incompatibility.

To select which of the 64K banks will respond to the ALTZP and Auxiliary Bank Switches, you use some locations in the $C0 page. These "64K select" locations take advantage of features of the $C0 page which the RAM card manufacturers have to hope won't get used for other things. There are two available locations on the $C0 page which send special signals up onto the auxiliary card; one is Annunciator 3 ($C05E/$C05F) and the other is the Game Control Strobe ($C07x). As you might have guessed, two of the manufacturers switch with one of the locations, and one uses the other location.

The MicroMax and Applied Engineering Cards use the softswitch at $C05E/$C05F (AN3) as the 64K select line. There are two potential problems with this. First, you can only have a one or a zero on this signal line, so you can never use this scheme to switch 64K banks on larger cards in the future. A more immediate problem is that the AN3 location has another important function in the //e. When a special jumper is installed on the auxiliary slot card, the Apple's IOU uses AN3 to manage super high res graphics. The result is that you cannot use one of the 64K banks while the super hi res jumper is installed.

The Neptune card from Titan Technologies uses the Game Control Strobe at $C07x. This strobe is used a great deal in commercial programs for the Apple, but Titan is counting on the fact that their card only responds to a "write" to that location while most programs that use the strobe operate it with a "read" instruction. Further, although the strobe fires in response to any address between $C070 and $C07F, most programmers address it as $C070 out of habit. When a program does a "write" to $C071, the Neptune card grabs a byte off of the data bus. The byte can contain, in theory, any number between 0 and 255, and each number would refer to a different 64K bank. The Neptune card has three 64K banks which respond to a write instruction with a 0, 3 or 5 respectivèly.

The Applied Engineering Card was designed to respond to $C07x, and was then rewired at the factory to respond to AN3. There may be different versions out there, so if there's any doubt, you can check pin 6 on the auxiliary card. If it's not connected to anything, then the AN3 system is in use.

An additional problem with all of these RAM cards is that when you select a new 64K bank, you are moving new RAM into the display memory areas. To avoid trashing the display, these cards must either reserve chunks of RAM in each bank to maintain duplicate copies of the display data, or they must always select the primary bank except during VBL periods when the screen memory areas are not being scanned by the video system.

If these conflicts seem to be too much to worry about, the best choice may be to install a 64K card in the auxiliary slot in order to take advantage of the special graphics features and of some future enhancements of ProDOS which may use the auxiliary 64K. Then think about one of the D*/D-E-F RAM cards discussed in Chapter 25 if you want a large RAM disk (see Figure 7). The "solid state drives" discussed in Chapters 22 and 24 don't do any bank switching, and so are less likely to interfere with any complex uses of the bank switch system by ProDOS, but they provide fewer features.

Video Display Management

The Apple's video display system is discussed in detail in Chapters 5, 6 and 7. In review, the Apple's memory is shared by the 6502 and the Apple's video display generator. Each microsecond of Apple time is divided into two halves, one half for the 6502 and the other half for the video system. The video display generator can be thought of as a "TV camera" which scans a range of the Apple's memory in order to generate signals to a CRT monitor which depicts what is in the display memory. The 6502 stuffs information into the display memory during its half of a microsecond and the video system scans that memory during its half.

In the Apple II/II+ there are four display memory areas which the video system can scan (Text1/Lo Res1, Text2/Lo Res2, High Res1, High Res2) and these are mapped out in Chapter 21, Figure 26.4. There is also a "mixed mode" in which high res graphics are displayed on most of the CRT except for the last inch or so which displays a few lines of text. When the

scan is being done in one of the text screens, the video display generator must be told whether it should interpret the data as ASCII codes or as low resolution graphics pixels. The various II/II+ video modes are reviewed in Figure 26.9.

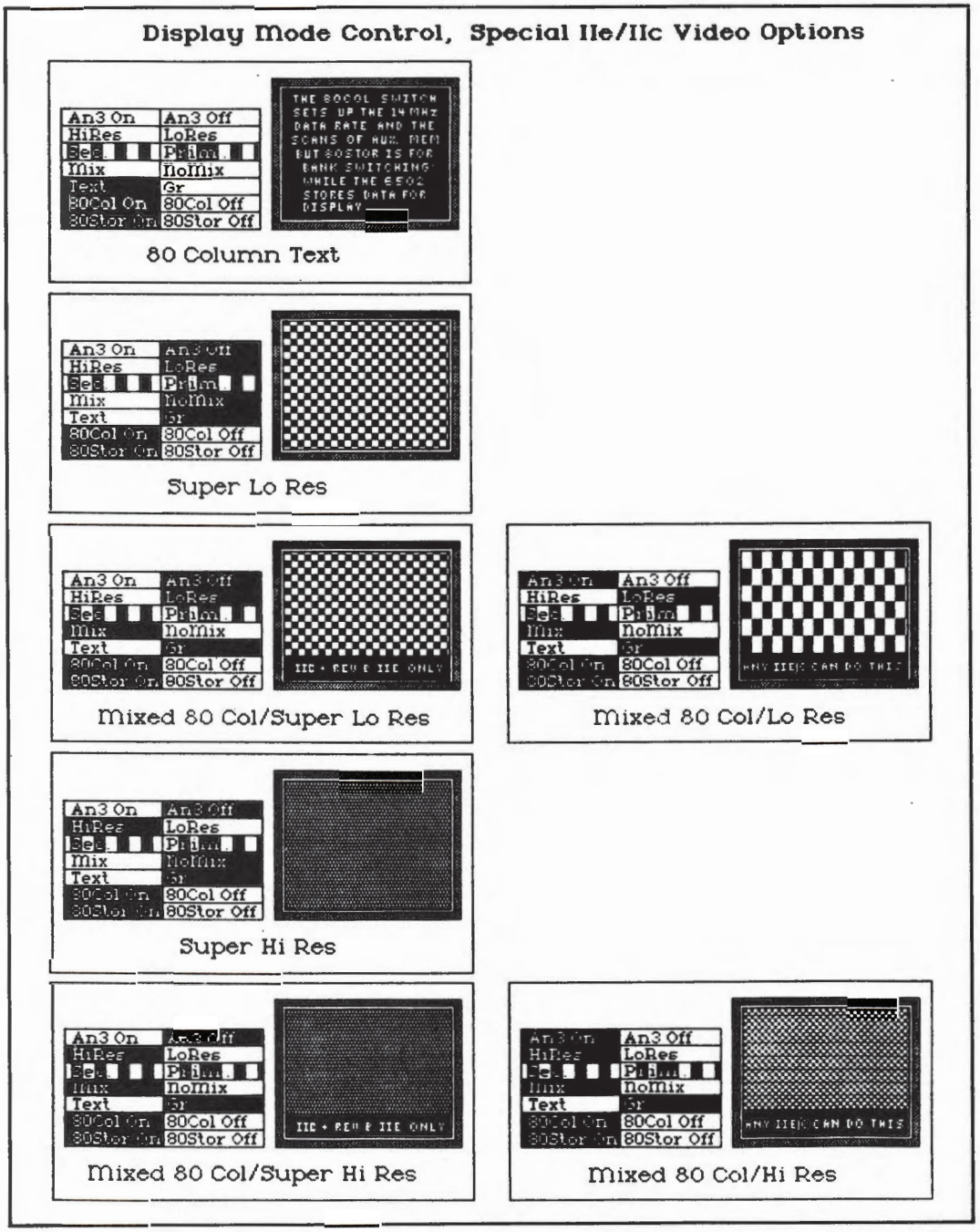


Fig. 26.9 Display modes and switch settings for the II/II+ and //e and //c. There are no 80COL or 80STORE switches to set on the II/II+. An3 (Annunciator 3) has no effect on the //e and //c video while 80COL is off.

II/II+ Video Display Control		
Hires	$C057	49,239
Lores	$C056	49,238
Secondary	$C055	49,237
Primary	$C054	49,236
Mix	$C053	49,235
NoMix	$C052	49,234
Text	$C051	49,233
Gr	$C050	49,232

Table 26.4 II and II+ video display control switches.

In the //e and //c, the scanning can be extended to include double width screens for text area 1 and for high res area 1. In the scanning process, the video generator alternates, reading one byte from auxiliary, one from main memory, back and forth as it sweeps out each scan line.

While scanning the text area and displaying ASCII codes as alphanumeric characters, the // e has the option of using two different character sets. Its standard character set emulates the Apple II/II+ except that it provides uppercase and lowercase. The alternate character set does not have flashing characters, but is able to display lowercase letters in inverse. In the //c, there is also a set of "MouseText icons" (see Figure 26.11) included in the alternate character set. These are also available in //e's with the appropriate set of new ROMs installed.

Video Display Switches

The video display switches affect the behavior of the video display generator, telling it where to scan and how to interpret what it sees. This must be distinguished from the effects of the 80STORE Display Memory Bank Switches discussed earlier. Those switches affect where the 6502 will put its bytes, rather than the way in which the video display generator looks for information. This is a two part system. First the programmer attends carefully to where data is put, then the video display generator is told where to look to find it.

The switches which control these functions are mapped out in Figure 26.1a, are listed in Tables 26.4 and 26.5, and are shown with resulting display configurations in Figures 26.9 and 26.10. Selection among the four possible screen display areas in the II/II+ is made by the use of two switches, "Lo-Res/Hi-Res" ($C056/$C057) and "Primary/Secondary" ($C054/$C055). The switch at $C052/$C053 ("NoMix/Mix") tells the video system whether or not it should scan in mixed mode, i.e., use graphics at the top and text at the bottom. Finally, the switch at $C050/$C051 ("Graphics/Text") tells the video system to choose between interpreting the text screen data as ASCII text characters or as lo res graphics blocks.

Additional Switches for //e and //c Video

When the //e or //c is operating in 80 column mode or displaying super high res graphics, part of the video display generator has to run at double its normal speed, and it has to manage the interleaving of data from the main RAM and auxiliary memory. This, incidentally, is not done by bank switching, because there is not enough time. The video display generator accesses both areas of RAM simultaneously but directs the data into special separate buffers. There is also a distinct "video data bus" which is managed by the video system (see above and Chapter 6 for a full explanation). To activate this complex behavior, you turn on a switch called "80COL" at $C00C/$C00D. This initiates the generation of 80 column text.

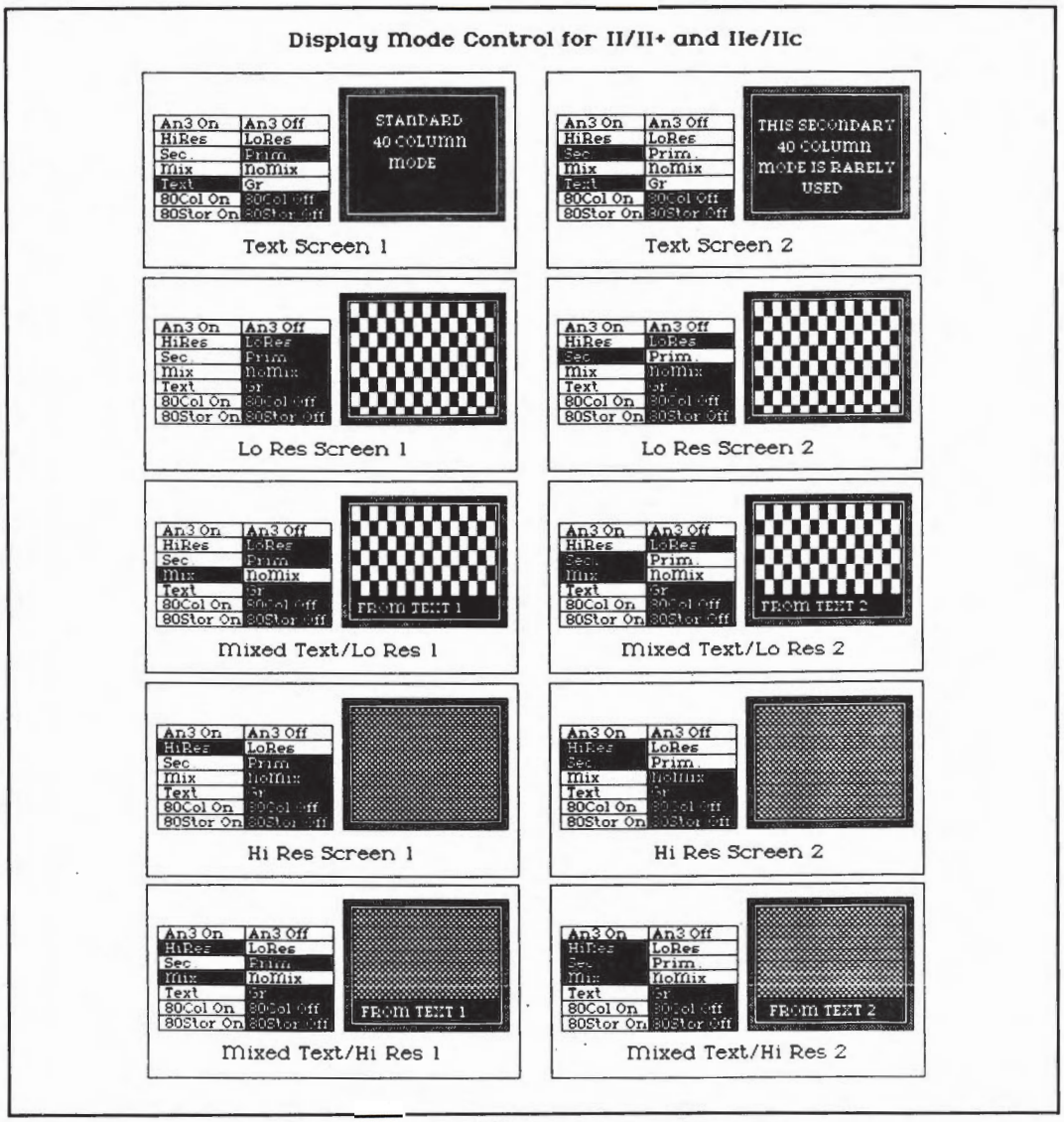


Fig. 26.10 Switch settings for the //e and //c display modes. Super hi-res and super lo-res require a Revision B or later //e with at least 64K of RAM in the auxiliary slot and the super hi-res jumper installed. AN3 (Annunciator 3) should be OFF for super hi-res and super lo-res graphics. For the //c, the AN3 function is marked NormRes/SupRes in Figure 1b.

IIe/IIc Video Display Control

Normal Graphics	$C05F	49,247	(IIc - must first set
Super Res Graphics	$C05E	49,246	IOUDIS on (W) $C07E)
Norm/Sup Status	$C07F	49,279	
Hires (SupTx)	$C057	49,239	
Lores (80Tx)	$C056	49,238	
HiRes Status	$C01D	49,181	
Secondary (Aux)	$C055	49,237	
Primary (Main)	$C054	49,236	
Page 2 Status	$C01C	49,180	
Mix	$C053	49,235	
NoMix	$C052	49,234	
Mixed Status	$C01B	49,179	
Text	$C051	49,233	
Gr	$C050	49,232	
Text Status	$C01A	49,178	
AltChr On	$C00F	49,167	
AltChr Off	$C00E	49,166	
AltChr Status	$C01E	49,182	
80 Col. On	$C00D	49,165	
80 Col. Off	$C00C	49,164	
80 Col. Status	$C01F	49,183	

IIe/IIc Video Ouput Status

GRline Status	$C077	49,271	(IIc 1 = graphics line, IIe 1 = text line)
VBL Status	$C019	49,177	(IIc Read and reset VBL IRQ)

IIe/IIc Display Memory Bank Switches

80 Store On	$C001	49,153
80 Store Off	$C000	49,152 (write only)
80 Store Status	$C018	49,176
Hires (SupTx)	$C057	49,239
Lores (80Tx)	$C056	49,238
HiRes Status	$C01D	49,181
Aux. Mem (Sec.)	$C055	49,237
Main Mem (Prim.)	$C054	49,236
Page 2 Status	$C01C	49,180

Table 26.5 //e and //c video display control and display memory bank switches.

To get super high res graphics, you need to do two things in addition to turning on 80COL. The first thing is to tell the video display system's address generator that you want it to look at hi res screen one instead of the text screen, and this is done with the "LoRes/HiRes" switch at $C056/$C057. The second step requires some additional circuitry which was not built into Revision A of the Apple //e (see Appendix D to find out if you have a Revision A motherboard).

Super High Res Circuitry in the //e

The Apple //e 80 column system is based on doubling the speed at which dots are sent out to the video screen. In 40 column mode, dots are sent at a rate of seven MHz, but in 80 column mode they are sent at 14 MHz. The result is that each dot is half as wide and that you get twice as many dots on a horizontal scan line (see Chapters 5 and 6). With twice as many dots, you can send 80 characters instead of 40, or in graphics, 560 dots instead of 280.

In the original design Apple did not plan to support super high res graphics, so they thought they had a problem to solve. If users turned on the video system while they were using 80 column text, they would get super high res, but be unable to plot dots in those screen columns whose data is stored in Auxiliary RAM (see above).

The way things work in a Revision A //e is that when you are in mixed mode hi res with 80 column on, the video output rate constantly changes from seven MHz to 14 MHz. In the top half of the screen, when graphics dots are being output, the IOU sends a "GR" signal to the PAL timing chip so it knows to slow the video data rate down to seven MHz. As the scan passes into the text area, the IOU turns off GR and the PAL kicks the video system up to 14 MHz to get out the 80 column text.

At some point shortly after the release of the Apple //e the folks at Apple decided that this wasn't really a problem at all. People should be permitted to use super high res if they wanted to, even if there was not currently any software or firmware to operate it. Rather than redesigning the IOU or the PAL, they came up with the simple solution of letting the "AN3" intervene by just diconnecting the GR line from the PAL. So, AN3 runs up onto the card in your auxiliary slot, crosses the jumper you've installed, and then runs back down to an unassuming 74LS 10 chip where it can disconnect GR (see Appendix A).

Super High Res Circuitry in the //c

The //c ciruitry for super high res graphics is based on the quick fix that was developed to give super high res to the //e. However, the various switches and signals have mostly been shrunk down into the inside of the GLU. The GLU has what amounts to the AN3 softswitch ($C05E/$C05F) built into it, although the switch is now formally called "DHires." The GR line from the IOU passes through the GLU on its way to the timing chip (TMG). Depending on the setting of the DHires switch, the GLU either sends its "TEXT" signal to the TMG (calling for 14 MHz video) or turns it off (to get seven MHz video).

If you do use the DHires switch, you should also be aware of the IOUDIS switch described in the mouse section of this chapter. As shown in Figure 26.1b, the $C05E/$C05F switch has an alternate set of functions, and IOUDIS must be on ($C07E) for the DHires function to work.

Addresses in the GLU

The GLU contains several softswitches and status report locations which have a slightly odd addressing pattern. If you examine the inputs and outputs of the GLU (see Appendix A), you'll find that it is connected to address lines A7, A6, A5, A4, A3 and A0 only. It gets a signal from the MMU that tells it if there is a $C0 page address it should be interested in, and then it can use address lines A7, A6, A5 and A4 to learn if it is, for instance, an $C07x or $C05x address, but then it only has A3 and A0 to find out which of 16 locations is being addressed.

The A3 address line tells the GLU if the location is either between $C070 and $C077 or between $C078 and $C07F. Then the A0 address line tells it which one of two adjacent addresses is being called. Thus, the $C07F/$C07E softswitch for IOUDIS is identical to using $C07D/

$C07C or $C07B/$C07A or $C079/$C078. This knowledge may clear up some discrepancies you find between addresses used in the firmware and addresses "documented" in reference literature on the //c.

Activating Super High Res

In a Revision B or later //e, you begin by putting a jumper on two pins on the auxiliary memory card, and this jumper connects the output from the AN3 softswitch at $C05E/$C05F to the GR killer. This gives your program means of tinkering with the timing of the video output and gets you super high res display. In the //c, you first enable the DHires switch by hitting IOUDIS at $C07E (49,278).

Undocumented Video Modes

There are a few odd and extraneous configurations of the //e and //c video system that have never made it into Apple's official documents. First, there is a super lo res mode that can be used in full screen or mixed text and graphics configuration with the switch settings shown in Figure 26.10. Many Apple owners were never quite sure what lo res was really for once the days of 4K RAM chips had been happily forgotten (see Chapter 21), and it's not clear what anyone would want to do with super lo res; but it's there.

An even deeper, darker, undocumented secret is the means for getting 40 column text with super high res. This is potentially useful for programs that wish to send mixed mode graphics to some users' television sets. The super high res gives you good control of color, while 40 column text won't overwhelm the TV's resolution. However, this "Super/40" mode is not too easy to work with, so it really is for true hackers and professionals only.

The Super/40 mode is based on an occasionally documented status signal on the //e and //c which we can call "GRline." This flag tells whether the Apple is currently doing a scan line in text or in graphics mode. Your program watches "GRline" and turns the 80COL switch off whenever GRline reports text output.

GRline can be read at $C077 in the //c, and the eighth bit is set for graphics but clear for text. Your program can stay in an endless check and toggle loop, and rely on VBL (and/or keyboard) interrupts to break out and take care of business.

In the //e, GRline can be read anywhere in $C07x, but its sense is opposite that of the //c—eighth bit 0 for graphics, 1 for text. You can't rely on interrupts, but you can rely on VBL status ($C019) staying on for the whole blanking period. Thus your program watches $C077 and $C019 to decide how it should set 80COL and whether it can do other things for a while.

Activating the Alternate Character Set

The last special switch for the //e and //c video generator is called ALTCHARSET. When it is flipped one way by a POKE 49166, 0 (a write to $C00E), you get the standard character set. Flipping it the other way with a POKE 49167, 0 ($C00F) makes it possible to get lowercase characters in inverse (see Table 26.6) and to get MouseText icons (//c or modified //e; see Figure 26.11). To see MouseText icons from BASIC, you have to invoke INVERSE display mode and then issue a PRINT CHR$(27).

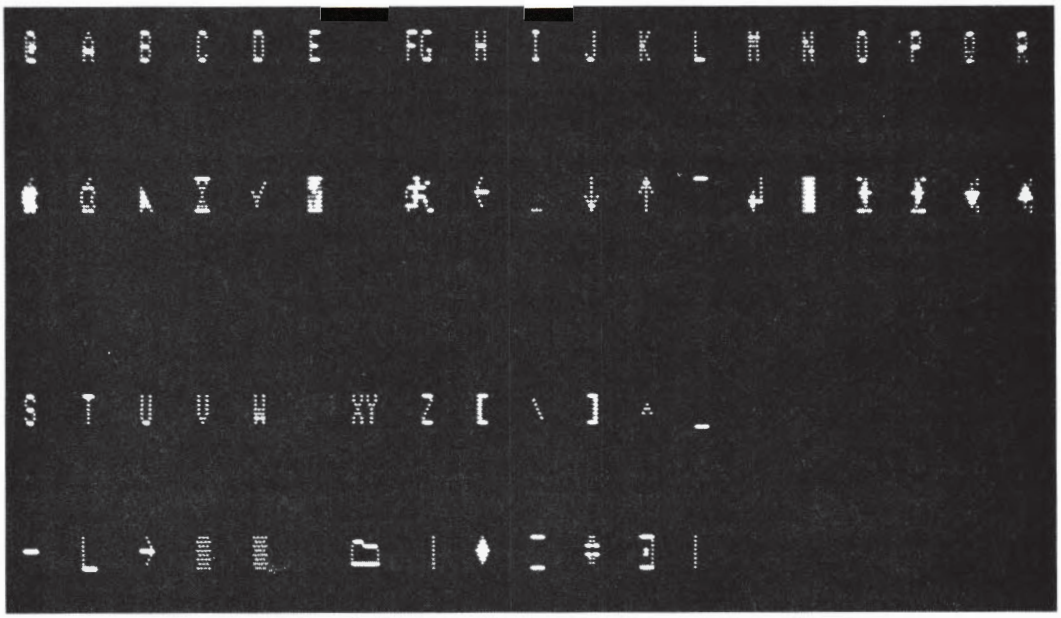


Fig. 26.11 MouseText Icon character set.

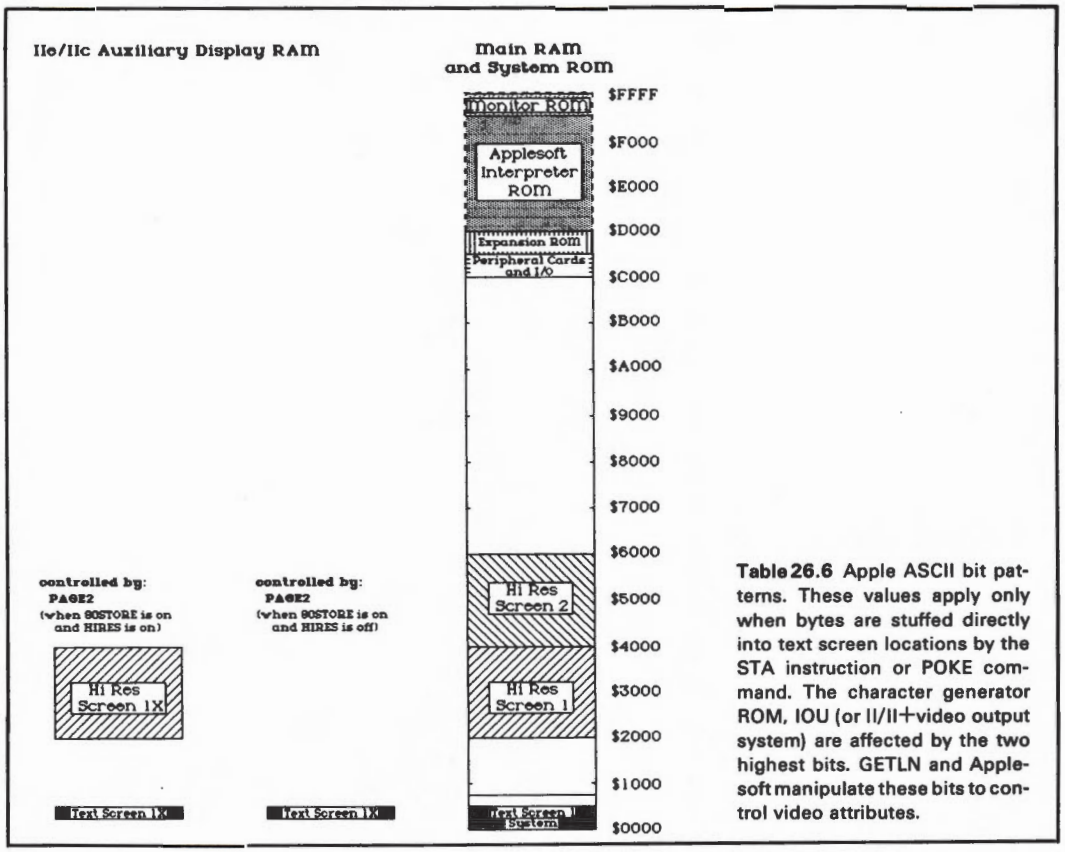


Table 26.6 Apple ASCII bit patterns. These values apply only when bytes are stuffed directly into text screen locations by the STA instruction or POKE command. The character generator ROM, IOU (or II/II+video output system) are affected by the two highest bits. GETLN and Applesoft manipulate these bits to control video attributes.

The //e and //c $C01x Status Report Panel

The various softswitches in the $C00x panel are all intended to be addresses by a "write" operation and the $C05x panel switches can be addressed with either a read or a write, but in either case, none of these locations is connected to the data bus. You therefore cannot go back to check how they've been set in case your program forgets. For instance, if there was an abrupt error, a program could catch the error and try to recover, but it might be necessary for the error catching routine to find out what was going on at the time of the crash. In the //c, these status reports are absolutely central to the function of interrupt driven operations such as are used for the mouse (see Chapters 9 and 27).

The //e and //c have been provided with a "Status Panel" at $C010 to $C01F (see Figure 26.1a). These locations are single bit flag inputs which report the settings of the various softswitches, as listed in Table 26.1.

In addition to softswitch reports, you can learn the status of the Any Key Down Flag (see Chapter 8), the D*/D-E-F bank switches (see Chapter 25) and the Vertical Blanking signal (see Chapters 5 and 9). In the //c the Vertical Blanking status functions differently than in the //e (see section above on mouse switches), and the SLOTC3ROM and SLOTCXROM inputs are replaced with inputs from the X-MOVE and Y-MOVE interrupt source flags (see Chapter 9).

Vertical Blanking has to do with the sweep pattern of the beam in the CRT tube of your monitor. A complete screen is drawn out 60 times a second, but at the end of each complete sweep the beam has to be snapped back up to the top of the tube to prepare for the next one.

This process takes a hefty 4.48 milliseconds in the Apple, which can allow for several hundred program steps. Those "steps in the dark" while the beam is turned off and snapping back up can be very important to programmers doing animation, moving a mouse cursor, etc. During the "vertical blanking period" they can move things around in display memory without causing any smearing or hashing on the video screen. It was possible for a programmer to identify the vertical blanking period by attaching special wires in the II/II+, but that was never very useful for commercial software. In the //e, the VBL input stays on for the whole blanking period. In the //c it turns on at the rising edge of an interrupt signal caused by VBL, but is cleared as soon as you read it at $C019.

//c Status Locations

The //c has a few more status report locations scattered around the $C0 page, mostly having to do with the mouse interrupt system. There are four inputs in the $C04x panel (see Figure 26.1b) which report on the setting of the four mouse interrupt softswitches in the $C05x panel. In addition, there are three status locations in the $C07x panel. One is used during mixed mode graphics and text display to learn whether text or graphics are being sent at any given instant. A second reports the setting of the DHires switch, and the third reports on IOUDIS (see discussion above of super high res activation).

Full Byte Data Ports

The built-in input/output system in the Apple II/II+ and //e has only one full byte data port, and it is for the keyboard. Unlike all the other built-in inputs, this is not just a flag input. In the //c, there are three additional full byte ports built-in: the disk drive port in the $C0Ex

panel, the printer port in the $C09x panel, and the modem port in the $C0Ax panel (see Figure 26.1c). The disk drive port is explored in detail in Chapter 23, and most aspects of the printer and modem ports are covered in Chapters 16, 17 and 18.

The Keyboard Input

The keyboard port actually involves several different functions. The principal and primary aspect of the system is the data port itself. Whenever a program "reads" address $C000 (49152) a byte of data from the keyboard is dumped onto the data bus.

The other parts of the system are concerned with helping a program figure out exactly when it should read $C000. In the original design of the keyboard port in the II/II+, a problem arose because the port continued to present the same data until a new key was pressed seconds or hours later. A program in the Apple would grab a new byte from the keyboard port and then come around to look again a few instants later and find the same byte sitting there. But did this mean that the person pressed the same key twice in a row or was it just the previous keypress data still sitting there.

The Keyboard Data Strobe Flag

To resolve this dilemna, the Apple uses a "Keyboard Data Strobe" and "Clear Keyboad Data Strobe" system. When a new key is pressed, the Ay-3600 keyboard chip (see Chapter 8) uses its data strobe line to set a single bit flag to 1. This is the Keyboard Data Strobe. A program is supposed to watch this Keyboard Data Strobe until it goes to 1. At that time, the program should hit $C000 and grab the new character from the keyboard. Immediately after the grab, and before doing anything else, that same program is required to hit the "Clear Keyboard Data Strobe" location at $C010 and thus clear the flag. Thus a new keypress is read once, and the program will not read from $C000 again until the Keyboard Data Strobe flag goes to 1 again.

The flag for the Keyboard Data Strobe is actually slipped into the eighth bit of the Keyboard Data port at $C000. Thus when you read a byte from $C000, the low seven bits contain the ASCII code for the character, but the eighth bit contains the status of the Keyboard Data Strobe flag. This two part input at $C000 must always be "read" (LDA or PEEK) since you actually mean to collect in some data. The Clear Keyboard Data Strobe location at $C010 will work with a read or write.

In the II/II+ the two locations are actually $C00x and $C01x since it doesn't matter exactly which of the 16 locations in the panel you use. In the //e and //c, however, there are other things going on in the $C00x and $C01x panels so you must be more circumspect (see Figure 26.1a and 26.1b). The safest approach is to always read from $C000 and clear with a write to $C010.

Any Key Down Flag

In the //e and //c, there is an additional kind of information provided by the Ay-3600 keyboard chip, and that is in the form of an "Any Key Down" flag. This flag is set as soon as the human hits the key, and it stays on as long as some key is still being held down (see Chapter 8). It is used internally in the IOU to generate the autorepeat feature, but it can also be read by a program.

Its location is $C010, exactly the same as the Clear Keyboard Strobe. In fact whenever you read the Any Key Down flag, you also activate the Clear Keyboard Data Strobe, but if you think through the sequence, you'll see that this is no problem.

Keyboard Interrupts in the //c

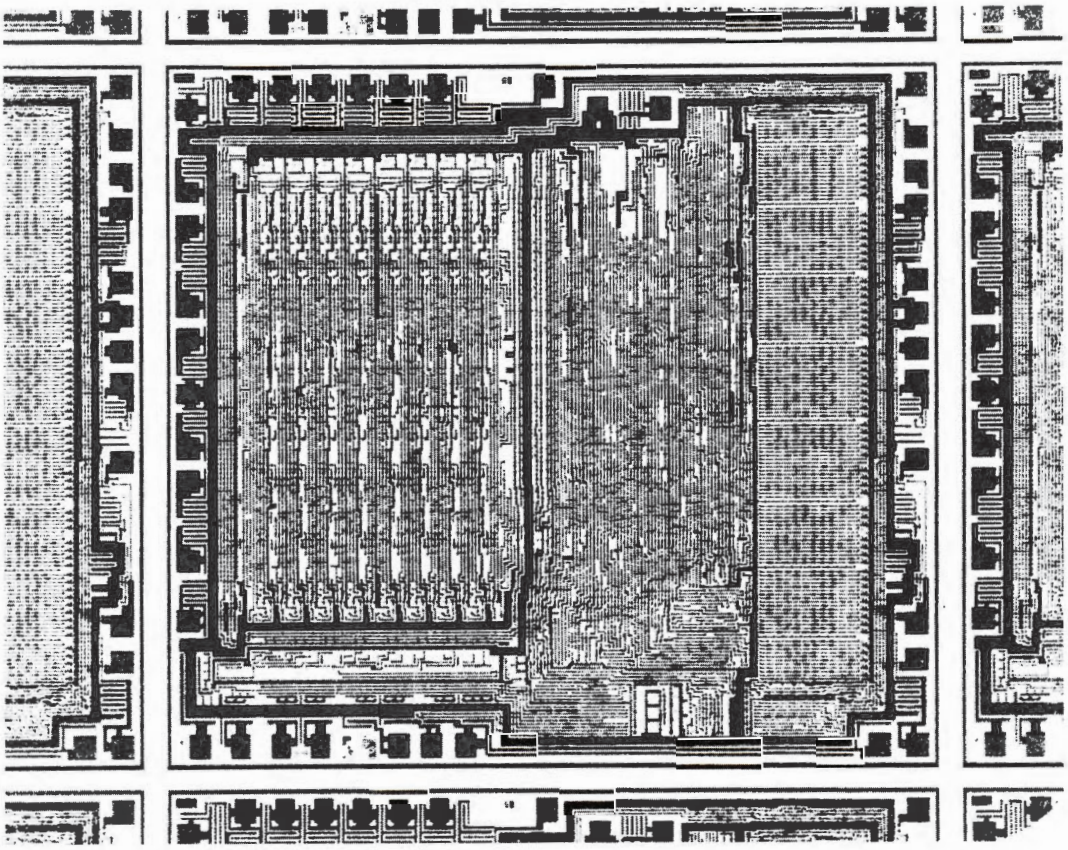
In the //c, there is an additional set of functions associated with the Keyboard Data Strobe and Clear Keyboard Data Strobe locations which have to do with interrupts. When the Ay-3600 sends its Data Strobe signal in the //c, it not only sets the Keyboard Data Strobe flag in the eight bit of $C000, but it also causes some activity in the serial modem port.

The Data Strobe line is connected to the Data Set Ready (DSR) pin in the 6551 chip that operates the modem port (see Chapters 16 and 18). The 6551 can be configured to respond to this by pulling the 65C02's IRQ (interrupt request) line, and thus abruptly seizing the 65C02's attention. This is a convenient way of letting the 65C02 go about its business without ever bothering to look at $C000 to check the Keyboard Data Strobe. If someone out there presses a key, an IRQ will occur, and the 65C02 can stop what its doing for a few moments, grab the new data byte, and go back to work.

The 6551 also records the presence of the Data Strobe signal by marking a 1 in the seventh bit of its "gister." When the 65C02 responds to its IRQ line, it will search through several "interrupt source flags" scattered around the $C0 page in order to figure out where the interrupt originated. It can check on the 6551's status register by looking at location $C0A9. If the seventh bit is set, then it knows a keyboard interrupt has occurred. It can then hit the Clear Keyboard Data Strobe location at $C010, and this will clear the interrupt status from the 6551.

PART 2

The Microprocessor Apple



Chapter 27

The 6502

A microprocessor is just one of those "hell bent" machines somewhat comparable to a tractor or lawnmower gone haywire or an unmanned wheat combine. Once you connect a five volt power source to its "Vcc" pin and attach it to its clock input, it just starts "processing" at a furious rate of over one million steps per second. This will happen even if it is just sitting there alone on a table top with none of its other pins connected to anything.

Proper etiquette calls for some benevolent soul to place a "branched list" of meaningful instructions in its path in order that it can be made to do something productive and useful. There are a small number of control inputs to the 6502, but most of these can accomplish little other than to make the thing change direction, i.e., start working on some other list. There is one control line called RDY (Ready) which can make it halt in its tracks, but this line is not connected to anything on the Apple's motherboard, and the only way it can be operated is by means of some smart device plugged into one of the Apple's accessory slots.

For the most part, a microprocessor is perfectly happy to blast along ferociously, running numbers through its own little built-in calculator, the Arithmetic Logic Unit (ALU), shuffling them among its various internal storage registers, pulling them out of its data buffer, shoving them into its address announcement system, and fiddling with its Read/Write (R/W) output control line. Fortunately, these machines can be reasoned with because they do include a set of about 150 tiny programs (appropriately called microprograms) which can be called up by name in meaningful sequences from outside the 6502.

This part of the book deals first with the internal ways and wiles of the 6502, then compares it with other microprocessors, and finally looks at how assemblers are used in attempts to effectively operate this machine. All of this also applies to the //c's 65C02 microprocessor, and the various new 65C02 features are covered throughout. In the next part of the book, there'll be a tour through the successful input/output subroutines in the Monitor and in various operating systems whose subroutines get used in just about every program ever run on the Apple (from homemade Applesoft calamities to highstrung thoroughbreds such as WordStar and VisiCalc).

The Microprograms

The striking photograph on the facing front page of this chapter is a blowup of a freshly made 6502 microprocessor. It is actually about a quarter of an inch square, and this photograph shows it surrounded by a few of the 50 or 60 identical companions fabricated in unison on the surface of a single circular, five inch silicon wafer. The large rectangle along the left edge is a kind of ROM which contains this 6502's set of microprograms.

INSTRUCTION ADDRESSING MODES AND RELATED EXECUTION TIMES (in clock cycles)

	Accumulator	Immediate	Zero Page	Zero Page, X	Zero Page, Y	Absolute	Absolute, X	Absolute, Y	Implied	Relative	(Indirect, X)	(Indirect), Y	Absolute Indirect
ADC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
AND	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
ASL	2	.	5	6	.	6	7	.	.	.	.	.	.
BCC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BCS	.	.	.	.	.	.	.	.	.	2**	.	.	.
BEQ	.	.	.	.	.	.	.	.	.	2**	.	.	.
BIT	.	.	3	.	.	4	.	.	.	.	.	.	.
BMI	.	.	.	.	.	.	.	.	.	2**	.	.	.
BNE	.	.	.	.	.	.	.	.	.	2**	.	.	.
BPL	.	.	.	.	.	.	.	.	.	2**	.	.	.
BRK	.	.	.	.	.	.	.	.	.	.	.	.	.
BVC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BVS	.	.	.	.	.	.	.	.	.	2**	.	.	.
CLC	.	.	.	.	.	.	.	.	2	.	.	.	.
CLD	.	.	.	.	.	.	.	.	2	.	.	.	.
CLI	.	.	.	.	.	.	.	.	2	.	.	.	.
CLV	.	.	.	.	.	.	.	.	2	.	.	.	.
CMP	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
CPX	.	2	3	.	.	4	.	.	.	.	.	.	.
CPY	.	2	3	.	.	4	.	.	.	.	.	.	.
DEC	.	.	5	6	.	6	7	.	.	.	.	.	.
DEX	.	.	.	.	.	.	.	.	2	.	.	.	.
DEY	.	.	.	.	.	.	.	.	2	.	.	.	.
EOR	.	2	3	4	.	4	4*	4*	.	.	6	5	.
INC	.	.	5	6	.	6	7	.	.	.	.	.	.
INX	.	.	.	.	.	.	.	.	2	.	.	.	.
INY	.	.	.	.	.	.	.	.	2	.	.	.	.
JMP	.	.	.	.	.	3	.	.	.	.	.	.	5

	Accumulator	Immediate	Zero Page	Zero Page, X	Zero Page, Y	Absolute	Absolute, X	Absolute, Y	Implied	Relative	(Indirect, X)	(Indirect), Y	Absolute Indirect
JSR	.	.	.	.	.	6	.	.	.	.	.	.	.
LDA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
LDX	.	2	3	.	4	4	.	4*	.	.	.	.	.
LDY	.	2	3	4	.	4	4*	.	.	.	.	.	.
LSR	2	.	5	6	.	6	7	.	.	.	.	.	.
NOP	.	.	.	.	.	.	.	.	2	.	.	.	.
ORA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
PHA	.	.	.	.	.	.	.	.	3	.	.	.	.
PHP	.	.	.	.	.	.	.	.	3	.	.	.	.
PLA	.	.	.	.	.	.	.	.	4	.	.	.	.
PLP	.	.	.	.	.	.	.	.	4	.	.	.	.
ROL	2	.	5	6	.	6	7	.	.	.	.	.	.
ROR	2	.	5	6	.	6	7	.	.	.	.	.	.
RTI	.	.	.	.	.	.	.	.	6	.	.	.	.
RTS	.	.	.	.	.	.	.	.	6	.	.	.	.
SBC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
SEC	.	.	.	.	.	.	.	.	2	.	.	.	.
SED	.	.	.	.	.	.	.	.	2	.	.	.	.
SEI	.	.	.	.	.	.	.	.	2	.	.	.	.
STA	.	.	3	4	.	4	5	5	.	.	6	6	.
STX*	.	.	3	.	4	4	.	.	.	.	.	.	.
STY**	.	.	3	4	.	4	.	.	.	.	.	.	.
TAX	.	.	.	.	.	.	.	.	2	.	.	.	.
TAY	.	.	.	.	.	.	.	.	2	.	.	.	.
TSX	.	.	.	.	.	.	.	.	2	.	.	.	.
TXA	.	.	.	.	.	.	.	.	2	.	.	.	.
TXS	.	.	.	.	.	.	.	.	2	.	.	.	.
TYA	.	.	.	.	.	.	.	.	2	.	.	.	.

* Add one cycle if indexing across page boundary
** Add one cycle if branch is taken, Add one additional if branching operation crosses page boundary

Table 27.1 Execution time in clock cycles for 6502 instructions. Very elegant machine language routines such as the RWTS routine that operates the disk drive rely on the exact timing of the various instructions. The effect of crossing page boundries (*) means that execution speed of a routine like RWTS will change if its starting location is moved. In order to assure complete software compatibility, William Mensch duplicated all of the timing anomalies when he designed the new 65C02 used in the //c.

These programs are identical in every 6502 ever made, no matter what kind of computer the 6502 is destined for, and they are copied very closely in the 65C02. The ROM which contains these microcodes is a major part of the "control section" of the microprocessor. The ultimate act of "processing" within the 6502 is the copying of a bit pattern from the microcode ROM to the latches which drive the output lines of the control section.

There are about 151 of these 6502 microprograms, and they vary in length from two on up to about seven steps (see Table 27.1). Each step of each program has a unique address within the microcode ROM. Each microprogram step is specified by a combination of two inputs to the ROM. The first input is an eight bit "opcode" which is held in the "instruction register." There could be as many as 256 different microprograms selected by the 256 possible eight bit opcodes, but the 6502A and the 6502B which are used in Apple II/II+ and //e computers have only 151 different microprograms (see Chapter 28). In the 65C02, that number is boosted to 178.

Within a microprogram, each step is identified by a "machine cycle number" between zero and seven which is provided by the "timing counter" (see Figure 27.1). When the 6502 is ready to begin working with a new opcode, it first clears the timing counter to "T0," and then fetches the opcode into the instruction register. Clock pulses arrive at the timing counter at a rate of about one megahertz (one million per second), and each pulse advances the counter by one (T1, T2, T3, etc.) and thereby identifies the next step in the current microprogram. The last step of each microprogram is to see that the next opcode is fetched and that the timing counter is reset to "T0".

Fetching Instruction Codes from the Address Space

In order for the 6502 to know which of its microprograms it should use and in what sequence it should use them, it must be able to read in an ordered list of supervisory opcodes. While these opcodes are waiting around not being used, they are stored outside of the microprocessor in ROM chips and in RAM chips, so it is quite crucial that the microprocessor have a simple and effective means for interacting with those external memory chips.

The system which manages interactions between the microprocessor and external memory involves two major parts. One part is responsible for figuring out exactly which single byte of information is needed. This first part might be called the "address preparation system." The second part is responsible for actually carrying out the electrical steps involved in copying the selected byte from its storage location in external memory to the appropriate place within the 6502, and it could be called the "memory I/O interface."

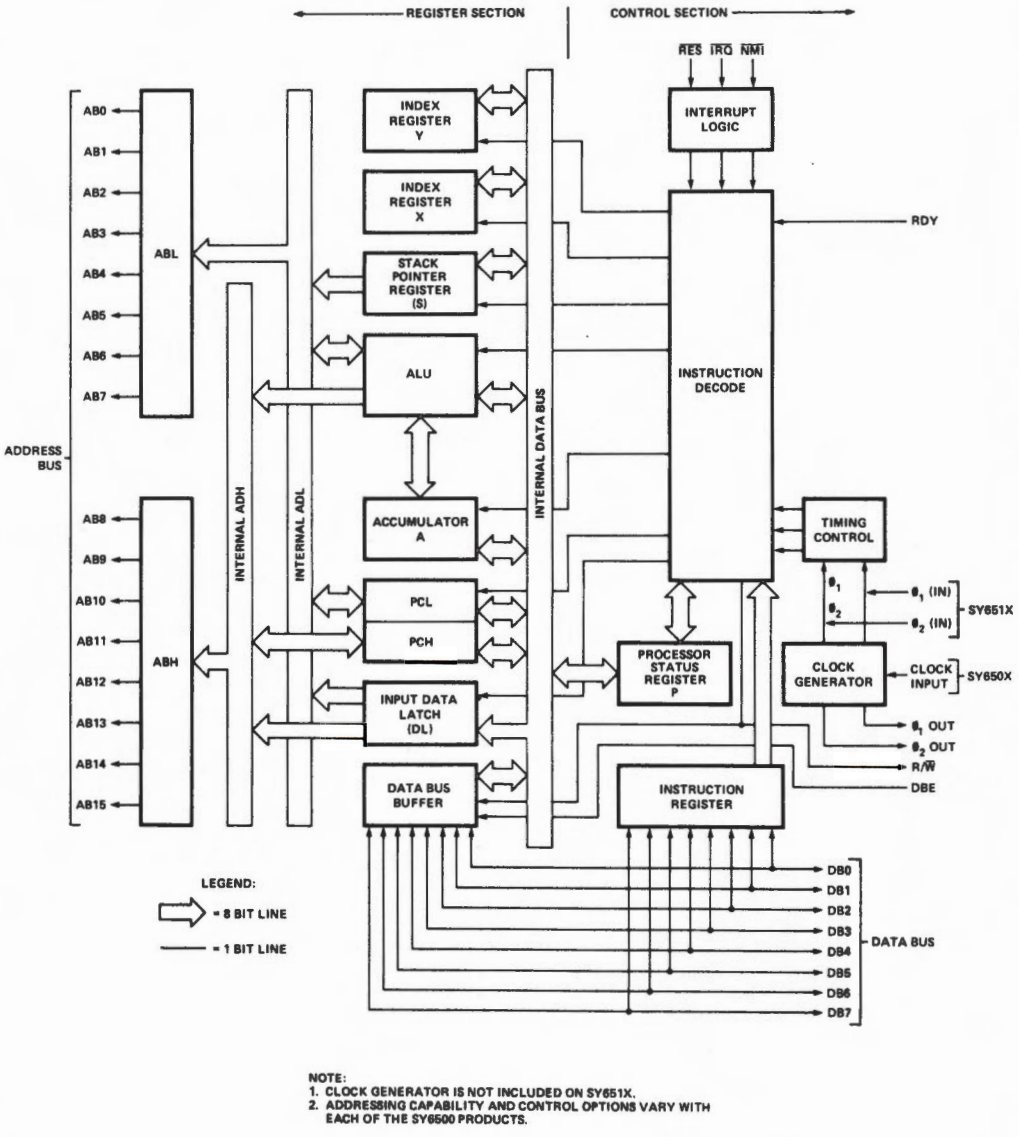
The Program Counter and the Address Space

At the heart of the address preparation system are two eight bit storage "registers" which are collectively called the Program Counter (PC). A great deal of the 6502's time and effort goes into controlling exactly what number is stored in the program counter. The program counter never contains instruction opcodes, and it never contains data for analysis. It exclusively contains the address of that one location in external memory where the 6502 can look to find its next instruction code.

Comments on the Data Sheet

The data sheet is constructed to review first the basic "Common Characteristics" — those features which are common to the general family of microprocessors. Subsequent to a review of the family characteristics will be sections devoted to each member of the group with specific features of each.

SY6500 Internal Architecture



NOTE:
1. CLOCK GENERATOR IS NOT INCLUDED ON SY651X.
2. ADDRESSING CAPABILITY AND CONTROL OPTIONS VARY WITH EACH OF THE SY6500 PRODUCTS.

Fig. 27.1 6502/65C02 block diagram.

Courtesy of Synertek, see page 2.

The nature and form of the address space of the 6502 is reviewed in great detail in Chapter 21, so a very brief overview will have to suffice here. This "address space" contains 65,536 locations. In the Apple, most of these locations can store one byte of information.

The address space is in fact a sort of figment of the 6502's imagination. The Apple has a variety of "bank switches" that can shift this theoretical space out onto peripheral cards, split it into pieces several inches apart, and otherwise cause it to be distorted and mangled (see Chapters 22, 25 and 26.) Throughout, however, the 6502 knows only that there is one perfectly straight and continuous stack of locations, whose addresses range from 0 to 65,535 (see Chapter 21, Figure 21.2.) It can place the address of any one of those locations into its program counter, caring little about which particular RAM, ROM or other addressable device is currently occupying that location.

For purposes of identification, the address space is divided into 256 regions called pages, and each page contains 256 locations (256 by 256 equals 65,536). In the program counter, one of the two registers (PCH for Program Counter High byte) keeps track of the page number, while the other register (PCL for Program Counter Low byte) keeps track of which location within the page is actually being addressed. Programmers often try to keep short lists of instruction codes entirely within a single page. In this fashion, they can see that the PCH is set once, and that all subsequent changes in the program counter can be handled by working with PCL alone.

Hexadecimal Numbers in the Address Space

Throughout Chapter 21, the locations in the address space are referred to in both hexadecimal and in decimal notation. The hexadecimal numbering system is reviewed in Chapter 21, but you can get along reasonably well without it. It is called hexadecimal because there are 16 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F, instead of the normal 10 digits. Whenever a hexadecimal number is used in most books, it is preceeded by a dollar sign ($) to avoid confusion with decimal numbers. However difficult it may seem to believe, the fact is that nearly everyone who spends much time fiddling around with addresses and memory locations tends to find hexadecimal much simpler to use than standard decimal.

To show how this makes things simpler, let's take page seven of memory and consider the addresses within it. You will recall that each page can have 256 locations. This is because the locations are marked out by the eight bits in the PCL register, and two to eighth power is 256. You should note that 256 is equal to $100 hex, and 255 is equal to $FF. In decimal, the first address in page seven would be $7 \times 256 = 1{,}792$, but in hexadecimal the first address would be $700. The last address in the page would be 1,792 plus 255 equals 2,047 in decimal, but a much clearer $7FF in hex. The next location in the space is at the beginning of page eight and its address is 2,048 decimal or $800 hex.

As a consequence of its binary soul, the 6502 thinks in something like hexadecimal. It is easier to carry on dialogue with the 6502 if you follow suit. Just think in nice round hexadecimal numbers and avoid trying to do conversions to decimal (see Chapter 21, Listing 21.1) unless it is absolutely essential that you do so.

Getting Addresses into the Program Counter

During standard operation, the program counter is loaded with the address of the location at which the next instruction code is stored. The 6502 then uses its memory I/O interface to

"read" the contents of that location. One popular way of describing this situation is to say that the program counter points to the next instruction. The interface looks where the program counter is pointing and grabs whatever's there.

As a result of this system, the 6502 never has to try to figure out what to do next. Rather, it constantly has to figure out where to look for its next instruction code. It will find out what it has to do as soon as that code is fetched. For the 6502 then, the key question at any given moment is: What number gets put into the program counter next?

The 6502 has three methods for managing the contents of the program counter. The first and simplest can be called "auto-increment" and involves a simple stepping along through the locations in a page. The second method uses program sequence instruction codes to modify the contents of the program counter. The third or "internal address" method is reserved for abrupt events which either initiate, interrupt or terminate the normal execution of a program, and it involves direct action by the 6502 to shove an address from its own internal ROM into the program counter.

Auto-Incrementing the Program Counter

The first method is a simple automatic approach designed for standard operation. This system assumes that the locations of the instruction codes will follow each other in strict sequential order. After each code is fetched, the contents of PCL are taken out, incremented by one, and stuffed back into PCL. This steps the pointer through the 256 locations in a page, one by one, calling in a "straight line" sequence of instruction codes.

When it's time to move into the next page, the system does a little two step. It first takes and increments PCL, but then it goes back and increments PCH as well, thus changing the page pointer. This is very important in "time critical" programs, because of the extra time it takes to increment across a "page boundary." An example of such a time critical program is the RWTS program that reads data from Apple Disk II drives and must stay in close synchrony with the spinning drive. Programmers who try to place RWTS into a new location in memory must be careful of exactly where they put it.

Program Sequence Instruction Codes

The second system is dependent on the contents of the instructions themselves. In the 6502 there are 12 program sequence instruction opcodes which can directly manipulate the contents of the program counter. These instructions are always at least two bytes in length. The first byte announces that a program sequence instruction has arrived, and the next one or two bytes tells what particular number should actually be put into the program counter. The first byte is called the opcode, and it is decoded to call up a microprogram. The succeeding one or two bytes are called extension bytes, and the programmer has substantial freedom in choosing what number will be described by this extension.

Relative Address Branches

Some of the program sequence instructions cause only a slight modification of the auto-increment system. Essentially, when one of these instructions arrives, it tells the microprocessor that it should change the program counter by an amount which is other than the usual one step increment. These "relative address" changes can be as large as 128 steps forward, but they can also decrement the program counter, moving it backwards by up to 128 steps. The selected amount is stored in the second byte of the instruction.

All of the 6502's relative address instructions are "conditional branches." This means that they first check some aspect of the internal status of the 6502 before forcing the branch. If the condition they are checking for is not met, then a simple one step auto-increment is performed. The nine different 6502 conditional branch instructions each watch for one of nine different internal states. In order to "force a branch," programmers often use an additional instruction to first set an internal state flag, and then immediately test that condition which they now know will be true. One of the time saving features of the 65C02 is a new Branch Always (BRA) command which is an unconditional branch that is always taken. This avoids the old two step necessary to get a forced branch in 6502 code.

Absolute Jumps and Subroutines

The simplest of the program sequence instructions is called Jump Absolute (JMP Abs). In an absolute jump, the current address in the program counter is abandoned completely. When the 6502 encounters this instruction, it knows that the two bytes immediately following it in memory are actually the two bytes of a new address for the program counter. It does two normal increments in order to read in those two bytes, but as they arrive, they are put into temporary storage and then stuffed directly into the program counter. The 6502 starts reading and incrementing again, but it may now be working in a very different area of memory.

A similar chain of events is caused by the Jump to Subroutine (JSR) instruction. The only difference is that the 6502 takes some actions which will later permit it to "remember" what was in the program counter just before the extension bytes got shoved in.

At first, a JSR proceeds just like a JMP Abs. The two extension bytes are located and brought into temporary storage. However, just before stuffing these bytes into the program counter, the 6502 takes a few extra steps. It copies the current contents of the program counter into its memory I/O interface section and then saves it in external RAM memory. Later on, when a Return from Subroutine (RTS) instruction is received, the 6502 is able to bring the old program counter value back in from memory, and thus get the program counter to point to an address just past the original JSR instruction. The JSR/RTS system makes it possible for a program to find its way back to where it started from after doing a jump.

The 6502 reserves a special area in its address space called the "stack," which it uses for temporary storage during JSR/RTS sequences. The stack is made up of the 256 bytes in the second page of memory ($0100 to $01FF) and the 6502 has a special register called the "stack pointer" which helps the 6502 keep track of where in the stack it left that return address. While the 6502 is doing a subroutine, it might encounter another JSR instruction. In that case it does another save to the stack. In fact, it can handle up to 128 JSRs in a row before hitting a single RTS. As the RTS instructions finally start showing up, it will pull the return addresses back off the stack on a "last in, first out" basis.

Indirect Jumps

There is one more variation on the jump instruction called Jump (Indirect) or JMP (Ind). This is actually a sort of double jump. When a JMP Abs instruction occurs, the extension bytes get stuffed into the program counter and the 6502 then goes about auto-incrementing through the new area of program, assuming that the first byte it encounters will be the opcode for a new and unrelated instruction. In a JMP (Ind) however, the first two bytes encountered at the new location are themselves effectively stuffed into the program counter and a second jump is performed.

Why not just do two regular jumps? Well, this gets into the subtle preferences of skilled assembly language programmers. In theory, this setup permits certain kinds of program loops

which carry out a series of jumps. At each turn of the loop, the indirect address is incremented so that a "table of jumps" get carried out. In fact, the 6502 has about 13 obscure and complex address construction systems. Some programmers delight in using them. Others stick to a subset of simple constructions.

By comparison, the popular Intel 8080 and Intel 8085 microprocessors just don't offer more than one or two addressing modes. The Z-80 adds one or two, but the Intel 8088 and 8086 tend toward simplicity. The skilled 6502 programmer can often express a rather complex relation of program paths in just six or eight bytes, while the same problem would require 30 or 40 bytes to define in 8080 machine language. The Motorola 68000 processor used in the Lisa and the MacIntosh offers over 100 different addressing modes, and it is proving to be very popular and easy to program among assembly language programmers.

Internal Addresses for Resetting the Program Counter

One problem with this smooth, flowing, looping instruction fetch system (auto-increment plus program sequence instructions) is that there is no way in and no way out. When the 6502 is first turned on, what will be in the program counter? Where will it get its first instruction from? Once it is running, this system provides no obvious means of breaking into a program to make the 6502 do something else for while. All programs end by looping back into some sort of repeating standby loop because if the 6502 ever comes to a dead end it effectively goes haywire, which either results in a completely unresponsive or "hung" computer, or, if you're lucky, hung but with a few cryptic notations on the screen about where it was and what it was doing when it got into trouble.

All of these problems are solved by the 6502's "interrupt system." There are four kinds of interrupts in the 6502, but what they all have in common is that the 6502 takes an address which it keeps stored internally in the microcode area, and shoves it into the program counter.

IRQ Interrupts

The gentlest of the interrupts is called Interrupt Request (IRQ). This is not a program instruction. Rather, it is the name of one of the pins coming out of the side of the microprocessor. Normally, the voltage on this pin is kept at 5 volts, but when, for instance, some peripheral device wants to stop the 6502 and have its own program run, it can cause an interrupt request by pulling this signal line down to 0 volts.

This is called a "request" because the 6502 does not have to respond. A programmer can send in a Set Interrupt Disable (SEI) instruction which tells the 6502 to ignore all IRQ signals. For instance, just before the Apple starts to write data onto a disk in the drive, it always uses an SEI to tell the 6502 to ignore IRQs. An interrupt while the disk directory is being written could destroy the directory, effectively destroying the disk. After the disk read, a Clear Interrupt Disable (CLI) instruction is issued to clear the interrupt mask and thus reenable the 6502's response to the IRQ line.

Assuming that the IRQ line is enabled, and thus the 6502 is set up to respond, it always finishes out the microprogram it is currently running, but then it acts somewhat as if it had just received a JSR instruction. It saves the contents of the program counter out onto the stack, and also saves the contents of its "status register" (to be discussed shortly). If this were a JSR, it would then shove the two extension bytes into the program counter, but in an IRQ situation it looks into its own ROM, and every 6502 ever made shoves a zero into the last bit in the PCL and a one into every other bit position in both parts of the program counter. This

results in the address $FFFE (65,534), or the next-to-last location in the address space. Then it launches into what amounts to a Jump (Indirect). It looks at the contents of $FFFE and $FFFF, and puts those contents into the program counter. Thus, the bytes in $FFFE and $FFFF contain the address for the computer's interrupt handling routines. When an IRQ is recognized the 6502 starts executing the interrupt handling routine stored in the computer's RAM or ROM.

Because of the SEI/CLI instructions, a device that wants to cause an interrupt never knows in advance whether or not the 6502 will respond. Thus the IRQ line is almost always turned on and left on until it is intentionally turned off later by the interrupt handling routine. This poses another problem when the 6502 does respond. Since the IRQ line is still active, why doesn't it interrupt the interrupt handling routine? To avoid the endless cycle that would result, the 6502 issues an automatic SEI instruction immediately upon recognizing the IRQ. It thus disables any future interrupts until it is ready to handle them. The interrupt handling routine must then reset or deactivate the source of the IRQ signal before a CLI instruction occurs. Thus a double interrupt is avoided.

In an Apple II+ or older //e, the contents of $FFFE and $FFFF are in motherboard ROM and they are always $40 and $FA. This tells the 6502 to do a jump to the program starting at $FA40 (64,064) (it's $FA86 in old Apple IIs). The purpose of that interrupt handling routine is to save the contents of a few more of the 6502's internal registers, and then to figure out where the interrupt request came from and turn over control to the originator of the request. When an interrupting program finishes its work, it can restore some of these saved contents to their original locations in the microprocessor and then issue a Return from Interrupt (RTI) instruction, after which the program it interrupted can resume as if nothing had happened. The RTI instruction causes an automatic CLI, thus permitting new interrupts to come it.

The $45 IRQ Bug in the Apple II/II+ and //e

That is how it works in theory. In fact, this old Apple II interrupt handling routine is not a successful machine language program. Apple had always warned peripheral manufacturers and programmers not to use IRQ. This bug was sufficiently serious that it was not fixed until 1984 when ProDOS was released. The problem is that the very first thing that happens at $FA40 is that the contents of the 6502's Accumulator register is stored at memory location $0045 (0069) in the zero page. This would not itself be a problem except that many other programs use location $0045, and most important among these programs is DOS.

DOS uses $0045 to help keep track of where it is supposed to get data for disk reads and writes. If an IRQ is handled, some otherwise irrelevant number from the accumulator gets stuffed into $0045, and when the interrupt service ends there is no way to restore the old contents of $0045. The next time DOS tries to write to the disk, it will get the wrong address from $0045 and disaster will strike in your disk data.

The ProDOS folks almost got caught by this bug as well. When this was realized, the final version of ProDOS was nearly ready for release and there was literally no space left in the ProDOS program to add a few bytes of code to fully solve the problem. To make space, the copyright notice included at $BF50 was shortened so an "AFTRIQ" routine and an "OLD45" location could be added.

Another important consequence of this problem has to do with bank switched memory cards (see Chapter 25). Many Apple's these days spend a lot of their time with RAM switched into the addresses in high memory which are normally occupied by the Monitor ROM. If an interrupt occurs while RAM is switched in, the forced jump through $FFFE will fail because

nothing meaningful will be there to guide the 6502 towards the interrupt handling routine. This is not a difficult problem to correct, but since interrupts have not usually been used in the Apple II, many bank switch memory manufacturers have not made any provisions for this.

//c Interrupt Handling and the //e Revision ROM

The arrival of the //c heralded a dramatic change in the role of interrupts in Apple IIs. After seven long years with a crippled interrupt system, the Apple II family has suddenly been transformed by the introduction of an elegant, complex, interrupt driven machine, the //c. Interrupts will never be as important in the //e as they are in the //c, but there is also now a revised version of the //e ROMs which corrects the old $45 interrupt bug.

The //c uses an array of interrupts for operating its mouse, serial ports, keyboard, and even for interacting with coprocessors in external expansion modules. The details of //c interrupt handling are covered in Chapters 9, 18 and 26. The changes involve not only an enhanced ROM interrupt handling routine, but a variety of hardware changes throughout the machine.

The //c Interrupt Handler program (which begins at $C803) corrects the $45 bug (see above) by storing the accumulator on the stack rather than in location $45. This change alone means that interrupts will work with DOS and Pascal programs as well as with ProDOS. The other major improvement at the system level is that it is fully integrated with the Apple's numerous RAM bank switching schemes.

Bank Switch Handling

To integrate with all the bank switching, the Interrupt Handler must handle two distinct problems. The first problem has to do with the validity of the system interrupt vector at $FFFE and $FFFF. When D*/D-E-F bank switching is being used (see Chapter 25), there will be RAM switched into those address locations when the IRQ occurs. To avoid any confusion, there is an intialization routine in the //c Monitor which copies the $C803 address from $FFFE/$FFFF in the Monitor ROM into that address in both the alternate and the main F-RAM. Thus, no matter what is switched in when the interrupt occurs, the 65C02 will be able to find its way to the interrupt handler at $C803. Further, since $C803 is in the //c $C000 space, the interrupt handler itself will never be bank switched out of the 65C02's address space when an interrupt occurs.

With program control securely handed over to the Interrupt Handler (IH) at $C803, a few more important points of system housekeeping must be attended to. The IH routine scans through all of the //c's various softswitch status report flags and checks the current setting of all the various auxiliary, alternate, and D*/D memory banks (see Chapters 25 and 26). It collects up all this information in a "memory configuration byte" which it stores in the stack and then goes about looking for the source of the interrupt.

User Interrupt Vector

The IH routine then allows two different levels of interrupt handling. It can handle many kinds of interrupts "transparently." This means that it services the calling device and then returns to the original, interrupted program but without informing the program that anything has happened.

The other level of interrupt handling involves various degrees of cooperation between the IH routine and a special interrupt handling routine included in the applications program that is running. If and only if that applications program asks it to, the IH routine will finish with all its most vital chores and then jump through the page $03 Interrupt Vector at $03FE and $03FF, which should point to the part of the applications program that handles interrupts.

Once everything is finished and it's time to return to the part of the running program that was interrupted, the IH routine pulls its "memory configuration byte" back off the stack, resets all the bank swithces, and then does an RTI.

Interrupts with the Revised //e ROM

The revised //e ROM (available after summer 1984) is modified both to solve the $45 problem and to integrate better with the memory bank switching system. The $45 bug is corrected as in the //c, but the bank switching problem is a little more difficult.

The principal problem in the //e is that there is no safe permanent ROM in which to store the Interrupt Handling routine itself. The closest thing to permanently installed ROM in a //e is the video firmware in the internal $C300 ROM. Therefore, the new ROM begins its interrupt handling in the Peripheral Card ROM Space for slot 3. Once this new interrupt handler is activated, it can set up a memory configuration byte just as in the //c.

The $C300 space entry will usually be available for interrupts, unless the //e owner is using an 80 column card in slot 3 instead of in the auxiliary slot (see Chapters 22 and 26). The Ultraterm slot 3 video card from Videx is also available in a revised form so that its own ROM has the new //e interrupt handling entry.

The Break Instruction

An interrupt can also be generated from software by including the opcode $00 also called Break (BRK). BRK differs from IRQ in that it cannot be ignored. However, other than that, it follows exactly the same sequence through to $FA40. One of the first tasks of the interrupt handling routine is to figure out whether the process was kicked off by an IRQ or by a BRK and to act accordingly. The BRK instruction is usually used only by assembly language programmers who are trying to locate a bug in a program. By putting a break at a strategic location, the programmer can halt the program and check on the exact status of all registers, etc. at that point in the program.

The BRK handling routine in the Apple II always writes out program information on the screen and brings you up into the Monitor command level with its asterisk cursor. You see the address where execution stopped and the contents of the Accumulator, X-register, Y-register, Processor Status register and Stack Pointer written out on the screen as follows:

```
0302- A=98  X=A8  Y=35  P=30  S=E8
```

To see this happen, type: POKE 768,0, hit Return and then type: CALL 768. You have just put a BRK instruction into location 768 in memory, and then caused the 6502 to execute it.

NMI

The third type of interrupt is called Non-Maskable Interrupt (NMI). Like IRQ, it has its own special pin poking out of the 6502, but it differs in three important respects. First, as its name indicates, there is no way for a running program to prevent the interrupt from occurring. Second, the 6502 resonds only to the initial signal edge so there's not any concern with double interrupts or setting and clearing masks. The third difference is in the addresses used in its "effective jump (indirect)." The initial address is $FFFA, a few bytes below the IRQ and BRK vector. The address stored in $FFFA, $FFFB is also quite different in that it points to a location in the fourth page of RAM, $03FB. In the IRQ/BRK interrupt, the 6502 usually jumps directly to a program in the monitor ROM, but since an NMI jumps to RAM, a programmer has fairly complete control over what happens at the outset.

Normally, $03FB contains another jump back up into the ROM to simply start the Monitor running with no special attempt at handling the interrupt. However, by first storing a different jump instruction at $03FB and then hitting NMI, the 6502 can be made to start running any program you choose. None of this applies to the //c in which the NMI line is permanently disabled.

NMI Cards for Software Backup

You might doubt that any one location in Apple memory could be the subject of bitter arguments, lawsuits and raging controversy, but the NMI vector at $FFFA and $FFFB is a good candidate for such because it is the key to the newest battlefront in the continuing war between copy protectors and copy protection crackers.

Most copy protected software is designed to prevent you from halting the program. For instance, pushing the reset (CTRL- Reset) key often has no effect. This is done so that the program cannot be examined or copied. Thus, for many years, crackers focused on copying the disks themselves. More recently, Dark Star Systems (Snapshot II), East Side Software (Wildcard), Micro-Analyst (Replay II), Central Point (Alaska Card), and Pirates Harbor (Crack-Shot), have been selling huge numbers of cards which use the NMI line to make copies of copy protected software. There is also a printer buffer from Interactive Structures and a 128K RAM card from Abacus (see below) which can be equipped with NMI copying capabilities.

Business oriented software on the CP/M operating system has never been copy protected and many of the most successful software publishers market unprotected CP/M software. However, the Apple DOS software market is very different, and small innovative program publishers have often relied on copy protection for economic survival. Nonetheless, when a CP/M user buys software, a few copies are usually made for everyday use, and the original is locked away in safety. Many legitimate purchasers of DOS software have been interested in these NMI cards in order to make backup copies for daily use.

If you decide to buy one, you should be forewarned that some of them are extremely difficult to use, and that some of them can be defeated by the protected software. Some of the cards require that you first run a program which changes the jump at $03FB. You are then provided with a switch or a button which pulls the NMI line low while the program is running. Program execution stops, the 6502 jumps to $03FB, and the copy software siezes control. This is easily defeated by a protection scheme that makes its own change at $03FB.

More complex cards catch the interrupt by detecting the first call to $FFFA, but this is also vulnerable to defeat, and so the race goes on.

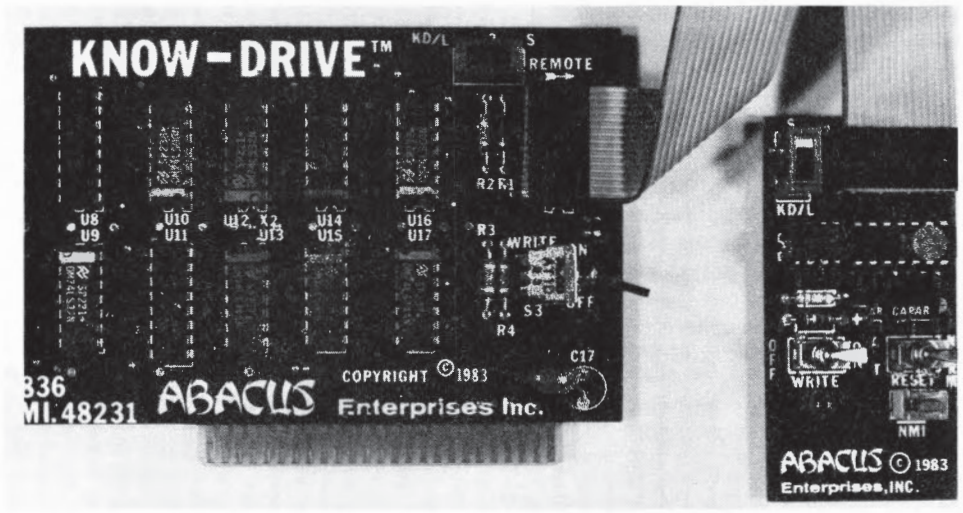


Fig. 27.2 Abacus Know-Drive NMI switch.

Using NMI for Background/Foreground Operation and Screen Grabs

The 128K RAM card from Abacus (see Figure 27.2) uses the NMI line for a much more legitimate and useful task. Abacus offers "Back to Back" software which permits you to have two completely separate programs loaded into the Apple with an NMI-based switch to throw you back and forth between them. For instance, VisiCalc might be loaded and running, but when you activate the Back to Back feature, the complete contents of the Apple memory and 6502 internal registers gets copied out to the RAM card. Now you can load and run, for instance, a game, and use the NMI-based feature to halt the game and snap back into VisiCalc, and then back into the game, etc. The Abacus Know-Drive also functions as a standard 128K RAM extension or as a RAM based "disk emulator" (see Chapter 25).

A second new and completely legitimate use of the NMI line is for Screen Grab printer interface cards, such as the Print-It from TexPrint (see Chapter 18). At any time while a program is running, you hit the button, the Apple stops what it is doing, and programs in ROM on the card go out and copy the text or graphics from screen memory out to your printer.

The Apple Reset System

The fourth and final kind of 6502 interrupt is called "Reset." Like NMI and IRQ, this interrupt has its own pin on the side of the 6502, and in the Apple this is connected to a key on the keyboard, also called reset. When a reset input comes into the 6502, it stops everything it is doing for six full clock cycles, and then it stuffs $FFFC into the program counter and launches into an indirect jump. This differs from IRQ and NMI in that no attempt is made to first save the contents of the program counter.

The reset line gets pulled automatically by a timer circuit on the Apple motherboard whenever the power is turned on. This automatic reset function was not included in the very first version of the Apple and, modest as it sounds, its subsequent addition was an important first step toward making the Apple user friendly. The reason that the 6502 waits for six clock pulses to arrive before starting in with its reset routine is that it assumes that it might take at least that long for the system clock to reach a stable rate of output.

In the II+, //e and //c, the contents of $FFFC and $FFFD is the address $FA62 which holds the first of a series of JSRs to short initialization routines that make up the beginning of the reset routine. The old Apple II Monitor ROM takes a slightly different route, but does most of the same JSRs. The reset routine initially launches into a series of steps which prepare the video screen for regular text output.

In the //e and the //c, the routine now checks to see if the Open Apple or Solid Apple keys are held down. An Open Apple tells the routine to execute a forced coldstart (see below). A Solid Apple in the //e calls for the self-test routine, while in the //c, a Solid Apple with Open Apple calls for the RAM/softswitch exerciser.

In an Apple II or II+, or in a //e or //c with no Apple keys down, the routine now looks down into low RAM at $03F3 and $03F4 (1,011 and 1,012) to find out what to do next.

The key first question that the reset routine asks at $03F3 and $03F4 is whether there is meaningless garbage stored at these locations. The $03F3 location is supposed to hold part of the reset vector (see below), and $03F4 is supposed to contain a logical permutation of the contents of $03F3. If this relationship doesn't hold, then the Apple assumes that the location contains garbage.

If so, the routine knows that the Apple has just been turned on and that nothing reliable is available anywhere in RAM. If this is the case, it puts something meaningful into $03F3 and $03F4, writes "APPLE II" on the top of the screen, and tries to turn on the disk drive. This procedure is called a "cold start."

This cold start sequence is also executed by a //e or a //c if the Open Apple key is down, no matter what is in $03F3 and $03F4. The cold start routine in the //e and //c also pauses to actively stuff garbage into hundreds of places in memory in order to destroy and thus protect any copy protected software currently loaded in RAM.

If, on the other hand, the routine finds appropriate codes already present in $03F3 and $03F4, it assumes that some program is supposed to be running, and it attempts to restart that program. Normally this means turning on the Applesoft Interpreter, but the way this is done is to look down at $03F2 and $03F3, and to do an indirect jump to the address stored there. This restart is called a "warm start."

Usually the address or vector at $03F2 and $03F3 is an entry point to Applesoft. However, if a manufacturer of protected software wants to prevent you from breaking out of the program and copying it, the programmer can change the contents of $03F2 and $03F3 so that a reset just brings you back up into the undisturbed protected program.

This is the reason Apple provided the forced cold start with the Open Apple key in //es and //cs. One problem in the II/II+ was that hitting reset from within a crashed program often just warm started you back up into the crashed program. The only way out was to turn the Apple off and then on again and too much of that sort of thing is a bit rough on the hardware—the on/off switch, for instance, often wears out in Apple II/II+s.

One unfortunate side effect of the standard warm start routine in the //e and //c is that it sets up the video for a 40 character display, but unlike the ESC4 command, it does not do a careful job of setting up the 40 column display. If you were using 80 columns at the time that you hit reset, only every other letter will be stored in the 40 column display memory, and what you see tends to be a bit weird. If this really bothers you, you can contact Call A.P.P.L.E. to get hold of a program from their August '83 issue which "tames" the //e and //c reset key.

Using the RDY/SYNCH System to Control Program Execution

There are two more pins on the 6502 which can be used to alter the progress of the instruction fetch system. The RDY line can cause the 6502 to "mark time," not proceeding with its microprogram until the RDY is turned back on again. All microprocessors have a control of this type primarily to permit them to work with memory chips which respond too slowly for full speed operation. The memory device is able to hold down RDY until it has its data ready to go. There are no memory chips in widespread use which are too slow for the 6502, but as newer processors such as the 68000 get up into the range of 12 megahertz operation, a RAM chip has less than 100 nanoseconds to respond and only specially made high speed RAMs can work that fast.

The most important use of RDY in the Apple II, II+ and //e is for interactions with some kinds of coprocessor cards—most notably the Microsoft Z-80 SoftCard. Use of RDY by the SoftCard is explained in Chapter 29.

Another possible use of the RDY line in the //e only is to step the 6502 through the instructions in a program, one instruction at a time. To do this, you have to know when each new instruction is about to begin. Since the various instructions in a program require different numbers of clock cycles, it is essentially impossible to predict when to hit the RDY line. Recall, however, that the clock counter is set to zero at the start of each instruction fetch. The 6502 sends out a pulse on its "SYNCH" pin every time it sets the counter to zero to begin an instruction fetch. Hardware can be set up to detect the SYNCH signal and hit RDY every time it goes on. The SYNCH signal was not connected in the Apple II/II+, but in the //e it has been brought out to pin 19 in the expansion slots. The primary use for SYNCH/RDY is for critical testing of new hardware and for debugging a balky Apple //e. The //c offers no convenient access to either RDY or SYNCH.

The Memory I/O Interface

The internal circuitry of the 6502 works fine for moving bits around among NMOS circuits (see Chapter 13) a few hundredths of an inch apart, but the 6502 needs special interface circuitry to deal with the TTL devices (see Chapter 13), and the comparatively long signal lines between it and the memory chips on the motherboard. This interface system is used both for handling program sequencing tasks and for handling the actual execution of instructions as the 6502 does its work. A similar situation exists in the 65C02, which has internal CMOS circuitry but must deal with the //c's TTL chips.

The interface system may be called a "memory I/O" interface because, in the 6502, the I/O ports (see Chapter 26) are treated as if they were just memory locations. In 8080 and Z-80 processors, a special M/IO output line alters the effects of the interface to distinguish between memory accesses and I/O chores. There are often completely separate external data and address systems for I/O in systems built around the 8080, Z-80 or 8086.

The Address Buffer

For outgoing addresses, there are two eight bit "Address Buffers" called ABL and ABH for the low and the high address byte, respectively. Once an address is prepared in the program counter, the control unit of the 6502 is able to route the address from the PC, along an "internal

address bus," and into the address buffer (see Figure 27.1). Sixteen address pins run out of the 6502 from the 16 bits of the address buffer, and these can be connected to TTL devices on the motherboard.

The Data System

The data system is a little more complicated. When an instruction is called up by an address in the address buffer, the memory device that is storing the instruction loads the instruction byte onto the main data bus on the Apple's motherboard. The 6502's eight data pins are connected to the external main data bus on the outside, but, inside the chip, each pin has two connections.

If the timing counter is at T0, and the incoming byte is expected to be an actual instruction opcode, then the byte is routed directly into the instruction register for decoding. The extension bytes, however, follow a different route. During T1 and T2 fetches, the incoming extension bytes are routed through the 6502's data bus buffer, onto the internal data bus, and then captured by the input data latch (see Figure 27.1).

The route from the data latch varies depending on what kind of instruction the extension is part of. If it is a "relative address" instruction, then the byte must be forwarded to the arithmetic area to be added to the contents of the program counter. In JMP (Abs) instructions, the incoming data must go into the program counter directly.

The data bus buffer is more than a simple gate onto the internal data bus. It actually has three different configurations. The one just mentioned involves its input state. Most of the time, however, its chief function is to prevent any noise or extraneous information from getting onto the internal data bus. It is only shifted into an input condition for brief periods during the "phase one" half of the clock cycle when an actual data input is expected. Most of the time, it is kept in a "floating" state, in which it allows nothing to pass between the internal and external data bus. During the execution of some instructions, it is also required to act as an output buffer for transmitting data to memory devices. This output state is also limited to occasional phase one periods.

The third element of the memory I/O interface is a separate pin called Read/Write (R/W). This line is used to inform RAM memory chips about what they should do when they are addressed. If this line is set to read, then the chip will respond to the address by sending out the appropriate data. If it is set to write (low voltage), then the RAM chip will be prepared to sieze new data from the external data bus.

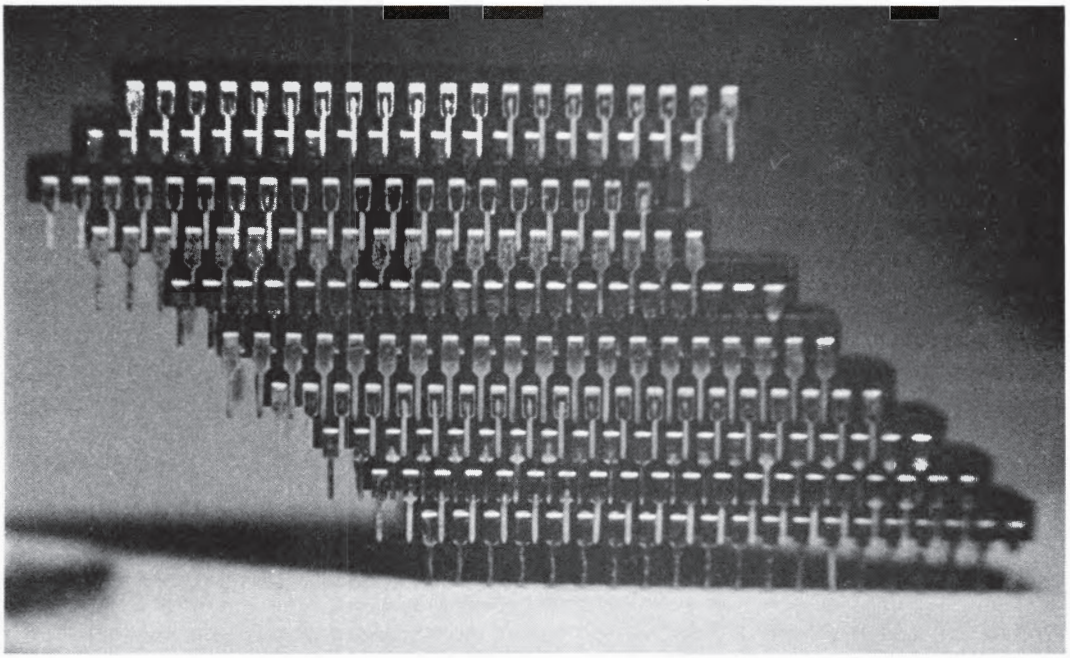
Direct Memory Access

Some peripheral cards for the expansion slots occasionally need to seize control of the Apple's memory for brief rapid data transfers. An interrupt is relatively slow and complex, so an alternative system called Direct Memory Access (DMA) is allowed. DMA is used by some high speed disk controllers (see Chapter 23), the Mountain Music Synthesizer (see Chapter 10) and by a variety of coprocessing cards including the Microsoft Z-80 SoftCard (see Chapter 29). During DMA, the Apple's clock is disconnected from the 6502, so it halts in the middle of whatever it is doing, and everything inside the 6502 freezes in place.

This is all done so that the external device can use the address bus and data bus to load data into or to copy data out of motherboard memory devices. However, it is not sufficient to just stop the 6502; the memory I/O interface must also be effectively disconnected. As long as the clock is cut off during phase zero, the 6502's data bus buffer will be set to float and so it is not a problem. The address buffers and the R/W line must actually be disconnected. This is done by passing these lines through buffer chips on the motherboard. The same DMA signal that stops the clock also floats the address and R/W lines by acting on the buffer chips (see Appendix A). The DMA device is then able to send its own addresses and R/W signals to the motherboard.

The //e distinguishes between data flow on the motherboard and data flow involving the peripheral cards. The accessory data bus which runs out to the cards is not always connected with the main data bus on the motherboard and auxiliary slot. The two buses are separated by an LS 245 three state buffer which must be set up correctly by the Memory Management Unit (MMU). The //e also has an alternate data bus which runs up to the auxiliary slot, and which is also separated from the main data bus on an LS 245 buffer chip. The DMA signal is routed to the MMU so that it will be sure to set up the proper relation among the three data buses during a DMA data transfer.

This extra buffer in the //e is included primarily just to add extra driving force to data signals going out to peripherals. A similar external data buffer is included on the II/II+, but it stands between the 6502 and everything else in the Apple, including the motherboard chips, so it is involved in all uses of the data bus. The II/II+ DMA line doesn't rely on the float state of the 6502's internal data buffer, but rather uses this external buffer to disconnect the 6502's internal data bus directly. None of the discussion on DMA applies to the //c because it does not have any circuitry to permit DMA.



Chapter 28

Execution of 6502 Instructions

Everything said about the 6502 to this point has been focused on the ordered delivery of instructions to the instruction register. To get a sense of the kinds of things the 6502 can actually do once it has been locked on to a proper sequence of commands, you need to browse among the instructions that have to do with the manipulation of data rather than with the management of program sequence.

These instructions fall into two broad categories; those that move data from one location to another, and those that can cause a little math or algebra. There are really just two commands for data transfers; "load," which means to move something into the 6502, and "store," which means to send it out. However, when you add in all the various modifiers you end up with a grand total of 42 different versions. The commands which deal with the Arithmetic and Logic Unit (ALU) contribute the remaining 93 instructions.

Data Transfer Instructions

Most of the work done by a microprocessor comes down to moving a byte of information from one place to another. In order to move a byte, the 6502 must be able to address both the source and the destination, telling the source to send out a byte and the destination to read it. However, the 6502, like all microprocessors, can only address one location in external memory at a time. Although the address buffer can be manipulated to point to any of the 65,536 locations in memory, the 6502 "sees" all of them as being brought into view for work at the one single 6502 data bus buffer (see Chapter 27, Figure 27.1).

Fortunately, the 6502 has been provided with three other storage locations on the microprocessor chip, and it is able to move data by enabling one of these internal registers and one address in external memory. To load data into the microprocessor, the 6502 control unit tells the internal register to read and tells the location visible in its data bus buffer to write. In order to store data from the microprocessor to memory, the roles are reversed. Therefore, the sequence of steps to move a byte from one location in external memory to a second location in external memory goes as follows: address first location, load the byte into the microprocessor, address second location, store the byte back out to its new home.

6502 Data Registers

Most data moves in the Apple take place among four locations; one is an address in external memory, and the other three are 6502 data registers. The most heavily used register for data moves is called the Accumulator (Acc or A). The two other data registers in the 6502 are called the X index register (X), and the Y index register (Y). The most common instructions

in any 6502 machine language program are LDA, which means to load the accumulator from a specified memory address, and STA, which means to store the contents of the accumulator out to some specified address.

The X and Y index registers historically reflect what could be called a "data counter," somewhat analagous to the program counter. However, unlike the program counter, the X and Y registers are occasionally used for other general storage purposes, and they are often ignored in the selection of a data address. One example of the use of the X register as part of a data counter would be the capturing of a series of keypresses as a line of stored text. A simple program to carry out this task might include the following steps as part of a loop:

```
LDA  keyboard (get byte from keyboard location and put it in the accumulator)
STA  $0200,X (now store the byte in the "line buffer" at memory address ($0200 plus the
                contents of the X register)
INX  (increment the contents of X by one)
```

Before beginning, a value of zero would be put into X, so the first byte from the keyboard would go into $0200. During the loop, the X register/data counter is incremented to one, so in the next pass the byte will get stored at $0201, the next pass would send it to $0202, then $0203, etc.

The three commands LDA, LDX and LDY are used, respectively, for shoving data from memory into the accumulator, X register and Y register, while STA, STX and STY send data in the other direction. One very efficient addition to the instruction set in the //c's 65C02 is an STZ instruction which shoves a zero into a location in memory. This is particularly handy for quickly clearing a screen. In the 6502, you would have to first do an LDA immediate to get a zero into the accumulator and then do an STA.

There is a special pair of instructions for moves between the accumulator and the 6502's stack on page $01 of RAM memory (see Chapter 21, Figure 21.4). You can "push" data onto the stack (PHA) or you can "pull" data off of the stack (PLA). A similar pair of instructions, PHP and PLP, can move the contents of the "processor status register" (discussed below) onto or off of the stack. The "stack pointer" keeps track of the actual destination or source for pushes and pulls. The 65C02 also has PLX, PLY, PHX and PHY instructions for moving the contents of the X and Y registers back and forth from the stack.

It is also possible to do internal moves among these three registers, and these moves are called "transfers." The commands involved are TXA, TYA, TAX and TAY. Notice that there is no TXY or TYX command. If you want to move a value from X to Y, you first do a TXA to get it from X into the accumulator, and then do a TAY to actually get it into Y. One other kind of shuffling involves the X-register and the contents of the stack pointer using TXS and TSX moves. However, they are considered extremely difficult to use correctly and so these two commands are avoided by all but the most daring and confident programmers.

Addressing Modes for Data Moves

Each transfer and stack command specifies both the source and the destination for the move. Data goes from one register to another within the 6502 or to a stack location held by the stack pointer. On the other hand, load and store commands need extension bytes to describe the memory location for the move. There are 13 different "addressing modes" in the 6502, and 15 in the 65C02. The various modes fall into three fairly straightforward categories.

Programming Characteristics

INSTRUCTION SET — ALPHABETIC SEQUENCE

ADC	Add Memory to Accumulator with Carry
AND	"AND" Memory with Accumulator
ASL	Shift left One Bit (Memory or Accumulator)
BCC	Branch on Carry Clear
BCS	Branch on Carry Set
BEQ	Branch on Result Zero
BIT	Test Bits in Memory with Accumulator
BMI	Branch on Result Minus
BNE	Branch on Result not Zero
BPL	Branch on Result Plus
BRK	Force Break
BVC	Branch on Overflow Clear
BVS	Branch on Overflow Set
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear Interrupt Disable Bit
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EOR	"Exclusive-or" Memory with Accumulator
INC	Increment Memory by One
INX	Increment Index X by One
INY	Increment Index Y by One
JMP	Jump to New Location
JSR	Jump to New Location Saving Return Address
LDA	Load Accumulator with Memory
LDX	Load Index X with Memory
LDY	Load Index Y with Memory
LSR	Shift One Bit Right (Memory or Accumulator)
NOP	No Operation
ORA	"OR" Memory with Accumulator
PHA	Push Accumulator on Stack
PHP	Push Processor Status on Stack
PLA	Pull Accumulator from Stack
PLP	Pull Processor Status from Stack
ROL	Rotate One Bit Left (Memory or Accumulator)
ROR	Rotate One Bit Right (Memory or Accumulator)
RTI	Return from Interrupt
RTS	Return from Subroutine
SBC	Subtract Memory from Accumulator with Borrow
SEC	Set Carry Flag
SED	Set Decimal Mode
SEI	Set Interrupt Disable Status
STA	Store Accumulator in Memory
STX	Store Index X in Memory
STY	Store Index Y in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
TSX	Transfer Stack Pointer to Index X
TXA	Transfer Index X to Accumulator
TXS	Transfer Index X to Stack Pointer
TYA	Transfer Index Y to Accumulator

ADDRESSING MODES

Accumulator Addressing

This form of addressing is represented with a one byte instruction, implying an operation on the accumulator.

Immediate Addressing

In immediate addressing, the operand is contained in the second byte of the instruction, with no further memory addressing required.

Absolute Addressing

In absolute addressing, the second byte of the instruction specifies the eight low order bits of the effective address while the third byte specifies the eight high order bits. Thus, the absolute addressing mode allows access to the entire 65K bytes of addressable memory.

Zero page Addressing

The zero page instructions allow for shorter code and execution times by only fetching the second byte of the instruction and assuming a zero high address byte. Careful use of the zero page can result in significant increase in code efficiency.

Indexed Zero Page Addressing — (X, Y indexing)

This form of addressing is used in conjunction with the index register and is referred to as "Zero Page, X" or "Zero Page, Y." The effective address is calculated by adding the second byte to the contents of the index register. Since this is a form of "Zero Page" addressing, the content of the second byte references a location in page zero. Additionally due to the "Zero Page" addressing nature of this mode, no carry is added to the high order 8 bits of memory and crossing of page boundaries does not occur.

Indexed Absolute Addressing — (X, Y indexing)

This form of addressing is used in conjunction with X and Y index register and is referred to as "Absolute, X," and "Absolute, Y." The effective address is formed by adding the contents of X or Y to the address contained in the second and third bytes of the instruction. This mode allows the index register to contain the index or count value and the instruction to contain the base address. This type of indexing allows any location referencing and the index to modify multiple fields resulting in reduced coding and execution time.

Implied Addressing

In the implied addressing mode, the address containing the operand is implicitly stated in the operation code of the instruction.

Table 28.1 6502 instructions, execution times, opcodes, number of bytes, available addressing modes, and effects on status flags.

Courtesy of Synertek, see page 2.

The simplest of the three categories doesn't use any extension bytes for addresses. The transfer, push and pull commands are said to use "implied" addressing because no extension byte is needed. The other simple form, called the immediate mode, actually carries data rather than an address in the extension. LDA immediate $25 takes the data from program memory and stuffs it into the accumulator without ever having to do identify a memory location to read from (in this case, putting $25 in the accumulator).

In the second category, all of the needed address information is found in the extension byte(s). The absolute and the indirect modes were introduced earlier (see Chapter 27) in reference to the JMP instructions. In a data move with absolute addressing, the two extension bytes following the opcode contain the address where the data can be found, while in indirect mode these two extension bytes only tell where to look to get that address. The relative mode used with branch instructions (see Chapter 27) fits into our category here, but it is not available for data moves.

The third category is called "indexed" addressing. This kind of addressing requires information from the extension bytes and also from the X or Y index registers. This is the kind of addressing used in the keyboard and line buffer example above. In an indexed mode, the contents of the X or the Y register is chosen to be added to the address in the extension bytes. The index can be added to an absolute address (Abs. Index X, Abs. Index Y) or it can be used together with indirect addressing. If you index the indirect address in the extension byte, it's called indexed indirect. If you first fetch the second address in the chain and then index that, it's called indirect indexed.

Thus, in our first category, we have two modes with no address in the extension byte (implied, immediate), our second category has three modes in which all the addressing information is included in the extension bytes (absolute, indirect, relative), and the third category has three modes which involve the index registers as well as the extension bytes (absolute indexed, indexed indirect, indirect indexed; see Table 28.2).

Several of them have zero page versions which require just one extension byte instead of two. These are more compact and execute more quickly, but most programmers treat these zero page modes as ways of managing simulated extra microprocessor registers. These zero page versions, as well as X and Y versions of some of the instruction modes bring the total to 13 in the 6502 and 15 in the 65C02.

Not all of the addressing modes are available for all of the instruction types (see Table 28.1). This is mostly a matter of convenience for the chip designers, since each new instruction type/addressing mode pair requires an additional microprogram, and space in the 6502 microprogram ROM is limited. The 65C02 contains several instruction type/addressing mode pairs not available in the 6502 (see Chapter 29, Table 1).

Math in the 6502

All of the mathematical data manipulations done by the 6502 can be thought of as moves in which the bytes pass through some part of the Arithmetic and Logic Unit (ALU) on their way to their destination. When a mathematical instruction arrives at the instruction register, the control section must be able to identify the source of either one or two pieces of data to be acted on, it must activate the proper portion of the ALU, and it must have a single destination address.

Relative Addressing

Relative addressing is used only with branch instructions and establishes a destination for the conditional branch.

The second byte of the instruction becomes the operand which is an "Offset" added to the contents of the lower eight bits of the program counter when the counter is set at the next instruction. The range of the offset is −128 to +127 bytes from the next instruction.

Indexed Indirect Addressing

In indexed indirect addressing (referred to as (Indirect,X)), the second byte of the instruction is added to the contents of the X index register, discarding the carry. The result of this addition points to a memory location on page zero whose contents is the low order eight bits of the effective address. The next memory location in page zero contains the high order eight bits of the effective address. Both memory locations specifying the high and low order bytes of the effective address must be in page zero.

Indirect Indexed Addressing

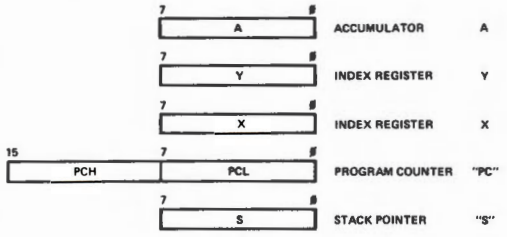
In indirect indexed addressing (referred to as (Indirect),Y), the second byte of the instruction points to a memory location in page zero. The contents of this memory location is added to the contents of the Y index register, the result being the low order eight bits of the effective address. The carry from this addition is added to the contents of the next page zero memory location, the result being the high order eight bits of the effective address.

Absolute Indirect

The second byte of the instruction contains the low order eight bits of a memory location. The high order eight bits of that memory location is contained in the third byte of the instruction. The contents of the fully specified memory location is the low order byte of the effective address. The next memory location contains the high order byte of the effective address which is loaded into the sixteen bits of the program counter.

Programming Characteristics

PROGRAMMING MODEL



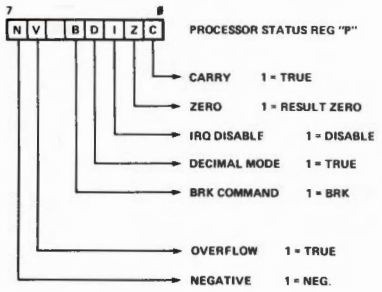


Table 28.2 Explanation of 6502 mnemonics and addressing modes. There are a number of additional instructions, and available addressing options on the 65C02 used in the //c.

Courtesy of Synertek, see page 2.

INSTRUCTION SET – OP CODES, EXECUTION TIME, MEMORY REQUIREMENTS

MNEMONIC	OPERATION	IMMEDIATE OP	IMMEDIATE N	IMMEDIATE #	ABSOLUTE OP	ABSOLUTE N	ABSOLUTE #	ZERO PAGE OP	ZERO PAGE N	ZERO PAGE #	ACCUM OP	ACCUM N	ACCUM #	IMPLIED OP	IMPLIED N	IMPLIED #	IND. X OP	IND. X N	IND. X #	IND., Y OP	IND., Y N	IND., Y #	Z. PAGE X OP	Z. PAGE X N	Z. PAGE X #	ABS. X OP	ABS. X N	ABS. X #	ABS. Y OP	ABS. Y N	ABS. Y #	RELATIVE OP	RELATIVE N	RELATIVE #	INDIRECT OP	INDIRECT N	INDIRECT #	Z. PAGE Y OP	Z. PAGE Y N	Z. PAGE Y #	CONDITION CODES N	Z	C	I	D	V
ADC	A + M + C → A (4) (1)	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3										✓	✓	✓	–	–	✓
AND	A ∧ M → A (1)	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3										✓	✓	–	–	–	–
ASL	C ← [7 0] ← 0				0E	6	3	06	5	2	0A	2	1										16	6	2	1E	7	3													✓	✓	✓	–	–	–
BCC	BRANCH ON C=0 (2)																														90	2	2								–	–	–	–	–	–
BCS	BRANCH ON C=1 (2)																														B0	2	2								–	–	–	–	–	–
BEQ	BRANCH ON Z=1 (2)																														F0	2	2								–	–	–	–	–	–
BIT	A ∧ M				2C	4	3	24	3	2																																M_7	✓	–	–	–	M_6
BMI	BRANCH ON N=1 (2)																														30	2	2								–	–	–	–	–	–
BNE	BRANCH ON Z=0 (2)																														D0	2	2								–	–	–	–	–	–
BPL	BRANCH ON N=0 (2)																														10	2	2								–	–	–	–	–	–
BRK	(See Fig 1)													00	7	1																									–	–	–	✓	–	–
BVC	BRANCH ON V=0 (2)																														50	2	2								–	–	–	–	–	–
BVS	BRANCH ON V=1 (2)																														70	2	2								–	–	–	–	–	–
CLC	0 → C													18	2	1																									–	–	0	–	–	–
CLD	0 → D													D8	2	1																									–	–	–	–	0	–
CLI	0 → I													58	2	1																									–	–	–	0	–	–
CLV	0 → V													B8	2	1																									–	–	–	–	–	0
CMP	A-M (1)	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3										✓	✓	✓	–	–	–
CPX	X-M	E0	2	2	EC	4	3	E4	3	2																															✓	✓	✓	–	–	–
CPY	Y-M	C0	2	2	CC	4	3	C4	3	2																															✓	✓	✓	–	–	–
DEC	M-1 → M				CE	6	3	C6	5	2													D6	6	2	DE	7	3													✓	✓	–	–	–	–
DEX	X-1 → X													CA	2	1																									✓	✓	–	–	–	–
DEY	Y-1 → Y													88	2	1																									✓	✓	–	–	–	–
EOR	A ⊻ M → A (1)	49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3										✓	✓	–	–	–	–
INC	M + 1 → M				EE	6	3	E6	5	2													F6	6	2	FE	7	3													✓	✓	–	–	–	–
INX	X + 1 → X													E8	2	1																									✓	✓	–	–	–	–
INY	Y + 1 → Y													C8	2	1																									✓	✓	–	–	–	–
JMP	JUMP TO NEW LOC				4C	3	3																													6C	5	3				–	–	–	–	–	–
JSR	(See Fig 2) JUMP SUB				20	6	3																																			–	–	–	–	–	–
LDA	M → A (1)	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3										✓	✓	–	–	–	–

MNEMONIC	OPERATION	IMMEDIATE OP	IMMEDIATE N	IMMEDIATE #	ABSOLUTE OP	ABSOLUTE N	ABSOLUTE #	ZERO PAGE OP	ZERO PAGE N	ZERO PAGE #	ACCUM OP	ACCUM N	ACCUM #	IMPLIED OP	IMPLIED N	IMPLIED #	IND. X OP	IND. X N	IND. X #	IND., Y OP	IND., Y N	IND., Y #	Z. PAGE X OP	Z. PAGE X N	Z. PAGE X #	ABS. X OP	ABS. X N	ABS. X #	ABS. Y OP	ABS. Y N	ABS. Y #	RELATIVE OP	RELATIVE N	RELATIVE #	INDIRECT OP	INDIRECT N	INDIRECT #	Z. PAGE Y OP	Z. PAGE Y N	Z. PAGE Y #	CONDITION CODES N	Z	C	I	D	V
LDX	M → X (1)	A2	2	2	AE	4	3	A6	3	2																			BE	4	3										✓	✓	–	–	–	–
LDY	M → Y (1)	A0	2	2	AC	4	3	A4	3	2													84	4	2	BC	4	3													✓	✓	–	–	–	–
LSR	0 → [7 0] → C				4E	6	3	46	5	2	4A	2	1										56	6	2	5E	7	3													0	✓	✓	–	–	–
NOP	NO OPERATION													EA	2	1																									–	–	–	–	–	–
ORA	A ∨ M → A	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3										✓	✓	–	–	–	–
PHA	A → M_S S-1 → S													48	3	1																									–	–	–	–	–	–
PHP	P → M_S S-1 → S													08	3	1																									–	–	–	–	–	–
PLA	S + 1 → S M_S → A													68	4	1																									✓	✓	–	–	–	–
PLP	S + 1 → S M_S → P													28	4	1																									(RESTORED)					
ROL	[7 0] ← C				2E	6	3	26	5	2	2A	2	1										36	6	2	3E	7	3													✓	✓	✓	–	–	–
ROR	C → [7 0]				6E	6	3	66	5	2	6A	2	1										76	6	2	7E	7	3													✓	✓	✓	–	–	–
RTI	(See Fig. 1) RTRN INT													40	6	1																									(RESTORED)					
RTS	(See Fig. 2) RTRN SUB													60	6	1																									–	–	–	–	–	–
SBC	A-M-$\bar{C}$ → A (1)	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3										✓	✓	(3)			✓
SEC	1 → C													38	2	1																									–	–	1	–	–	–
SED	1 → D													F8	2	1																									–	–	–	–	1	–
SEI	1 → I													78	2	1																									–	–	–	1	–	–
STA	A → M				8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3										–	–	–	–	–	–
STX	X → M				8E	4	3	86	3	2																															96	4	2	–	–	–	–	–	–
STY	Y → M				8C	4	3	84	3	2													94	4	2																	–	–	–	–	–	–
TAX	A → X													AA	2	1																									✓	✓	–	–	–	–
TAY	A → Y													A8	2	1																									✓	✓	–	–	–	–
TSX	S → X													BA	2	1																									✓	✓	–	–	–	–
TXA	X → A													8A	2	1																									✓	✓	–	–	–	–
TXS	X → S													9A	2	1																									–	–	–	–	–	–
TYA	Y → A													98	2	1																									✓	✓	–	–	–	–

(1) ADD 1 TO "N" IF PAGE BOUNDARY IS CROSSED
(2) ADD 1 TO "N" IF BRANCH OCCURS TO SAME PAGE
ADD 2 TO "N" IF BRANCH OCCURS TO DIFFERENT PAGE
(3) CARRY NOT = BELOW
(4) IF IN DECIMAL MODE Z FLAG IS INVALID
ACCUMULATOR MUST BE CHECKED FOR ZERO RESULT

X INDEX X
Y INDEX Y
A ACCUMULATOR
M MEMORY PER EFFECTIVE ADDRESS
M_S MEMORY PER STACK POINTER

+ ADD
– SUBTRACT
∧ AND
∨ OR
⊻ EXCLUSIVE OR
✓ MODIFIED

– NOT MODIFIED
M_7 MEMORY BIT 7
M_6 MEMORY BIT 6
N NO CYCLES
NO BYTES

Table 28.2 (continued).

Courtesy of Synertek, see page 2.

Although only one byte can move on the internal data bus in any given cycle, there is a special "Acc-ALU" bus which provides an alternate route from the Accumulator to the ALU. It is therefore possible to select one location in external memory and select the accumulator, and have the two of them simultaneously send bytes into the two input ports of the ALU. The result output either swings back around into the accumulator or gets sent back to the source location in memory.

The ALU in the 6502 can perform four kinds of basic mathematical manipulations: addition, complementation, shifting and logical transformation. Addition is just what it sounds like, and complementation alters a binary number so that you can simulate subtraction with the ALU's addition circuitry. Shifting can be used with addition to simulate multiplication and division, or it can be used with some of the logical transformations for various kinds of AND, OR, etc. bit fiddling.

All of the 6502's math is done with eight bit numbers between 0 and 255, so the ALU provides for chaining together these basic operations to work with larger numbers. The most important such provision is a "Carry" flag in the "processor status register." The carry bit in this register can be changed directly with the Set Carry (SEC) and Clear Carry (CLC) instructions, but it also responds automatically to certain mathematical operations. For instance, if you add $80 (128) plus $80 (128), you get $100 (256). The eight bit result of the addition is $00, but this operation will "set" the carry flag. A program can check the carry flag to determine the actual result of the operation.

Addition and Subtraction

The principal addition command is Add with Carry (ADC), and it always adds a value in memory to a value in the accumulator, then stores the results in the accumulator. The original value in memory is left unchanged. Like the LDA command described above, it has a variety of addressing modes ranging from immediate data in an extension byte to indexed indirect (see Table 28.1). There is also a Subtract with Carry (SBC) command which is very similar in operation.

Simple commands to increment (add 1) to the index registers (INX, INY) and to decrement (subtract 1) from the index registers (DEX, DEY) get by with implied addressing, but increment and decrement commands which can act on values in memory (INC, DEC) have some variety of addressing modes. All of the increment and decrement commands deliver their result somewhere other than the accumulator.

When an addition or subtraction is carried out, the ALU actually reports several aspects of the result to the processor status register. The carry has already been mentioned, but in addition there is a special flag to indicate if the result was zero, or if it was a negative number. There is actually a special subset of subtraction commands which do the subtraction, set the status flags, and then just abandon the results, leaving both the memory value and the accumulator unchanged. These are called "comparison" commands and they are very central to all machine language programs.

Using Comparison to Control Program Sequence

For example, if you take a glance at an ASCII Code table (see Chapter 18, Table 18.1) you will notice that any hex number lower than $20 is a "control character," $20 itself is a blank

space, and anything higher is probably a printable character. As each keypress comes in from the keyboard, a program could use a comparison command to find out what it should do. The flags which are set by the comparison subtraction can be monitored by branch commands (see Chapter 27) as follows.

The number $20 is loaded into the accumulator, then as each new character is typed, the program grabs it from external memory and does a CMP (address) to subtract it from the contents of the accumulator. If it's a control character, none of the flags will be set; if it's a space ($20), then the result of the subtraction will be zero and the Z flag (and the carry flag) will be set. If it's greater than $20, the carry flag will be set.

This CMP command will be followed immediately by a conditional branch command which will cause a branch if the carry flag is clear (BCC) to a control character handling routine. The next command will be a BEQ which will handle the blank space if the zero flag is set, and then finally a BCS command will catch any characters which are not less than $20 and not equal to $20. This last branch is a forced branch, since there is no condition in which it would not be taken.

Shifting and Multiplication

Since microprocessors work fairly quickly, it is not unreasonable to do multiplication via a large number of additions; however, shifting is far faster and more direct. The direct analogy to everyday math is the simple way of multiplying by 10 in decimal work. You just slide the number over to the left and put an extra zero on the end; 345 becomes 3,450. This is how shifting works. You can't get very far with shifting in decimal arithmetic since it doesn't work for multiplying by 2, 3, 4, 5, 6, 7, 8 or 9, but in the binary system there's nothing between 01 (which equals one) and 10 (which equals two), so shifting becomes a very effective way of doing multiplication.

Eight bit numbers can be shifted left for multiplication (ASL) or right for division (LSR). When you shove an eight bit number to the left, the last bit falls off the end. If you are going to need to know what this eighth bit is, you use a "rotate" instruction instead of a shift. In an ROL, that eighth bit gets put into the carry flag, and vice versa for the first bit in an ROR.

Logic Functions for Bit Fiddling

In the world of binary numbers, there is a collection of "logical operations" which play a very important role in mathematical work. In the Apple, these commands are important for gaining control over the setting of individual bits within a byte. There are innumerable places within the Apple where certain bits serve as "flags" to cause events or control choices, so the availability of machine language instructions to act on individual bits is crucial.

The three principal logical operations are AND, OR and EOR (Exclusive OR). They are used in various instructions which use one number from the accumulator and one number from memory.

AND:

If the acumulator bit is 1 AND the memory bit is 1, then the result will be 1:

1 AND 1 = 1

1 AND 0 = 0

0 AND 1 = 0

0 AND 0 = 0

OR:

If the acumulator bit is 1 OR the memory bit is 1, then the result will be 1:

1 OR 1 = 1

1 OR 0 = 1

0 OR 1 = 1

0 OR 0 = 0

EOR:

If the accumulator bit is 1 OR the memory bit is 1, then the result will be 1, but not if both are 1:

1 EOR 1 = 0

1 EOR 0 = 1

0 EOR 1 = 1

0 EOR 0 = 0

The machine language instructions work on an entire byte all at once, but they compare individual bits within the accumulator byte with the corresponding bits in the memory byte:

1001 1110

AND

0011 0110

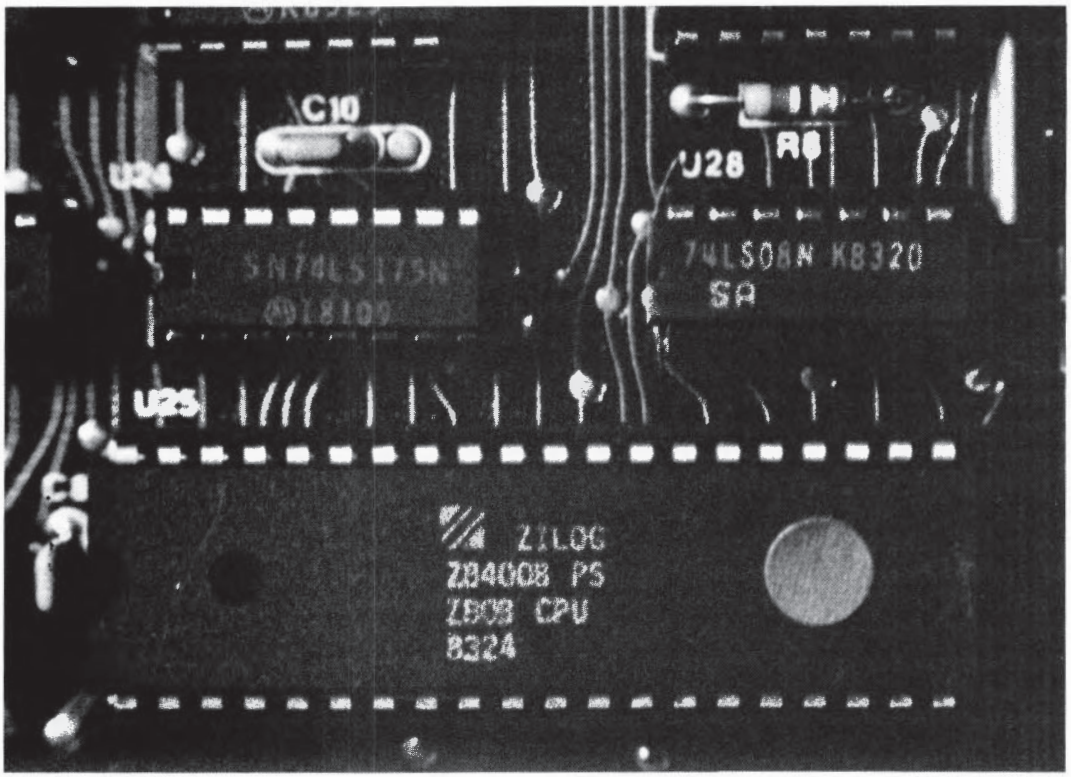
0001 0110

Thus, the machine language program can place a bit pattern in the accumulator and use it to selectively interact with particular bits in the target byte in memory.

In the 6502, there are four instruction that do logical bit operations. The AND, ORA (OR Accumulator), and EOR instructions do the three primary operations. In all three of these instructions, the result of the operation is left in the accumulator while the byte in memory is left unchanged. In addition, the operation may set various flags in the processor status register.

The fourth logic instruction for the 6502 is BIT. This is actually an AND instruction, but its distinction is that it leaves both the accumulator byte and the memory byte just as they were before the operation. All that changes are the flags in the processor status register. This is often used by Apple routines that must test the eighth bit of a byte in memory. The routine issues a BIT command and then does a "conditional branch" based on the result. If the eighth bit was clear, then a BPL instruction will work. If the eight bit was set, then a BMI instruction will work.

The 65C02 has two more logic instructions, TSB (test and set bits) and TRB (test and reset bits). The TSB instruction is like ORA except that it leaves the result in memory but the accumulator unchanged. The TRB instruction is a little trickier in that it does an AND with the inverse of the accumulator byte and then leaves the result in memory. The effect of a TRB instruction is to clear all bits in the memory byte unless the accumulator bit is 0 and the memory bit is 1.


C10
U24
U28
R8
74LS08AN KB320
SA
U25
ZILOG
Z84008 PS
Z80B CPU
8324

Chapter 29

The Microprocessor Families

Other Microprocessors for the Apple

One of the wonderful things about owning an Apple II is that if you don't happen to like the microprocessor you're using, you can go out and buy a card with a different one, shove it in a slot, and go on with you're work at a fraction of the cost of buying a new computer. This way, you can invest in cutting edge technology without the danger of getting sliced. If your souped up 32 bit 1985 microprocessor turns out to be unable to run a word processing program, well, you've still got a functioning Apple.

However, although it is very nice to be able to have all this freedom of choice, microprocessor cards can be fairly expensive, and there are some very important things you should consider if you're thinking about getting one.

The most important point to make here is that no matter how wonderful a given microprocessor may be in theory, the card on which it is mounted and the software that comes with it will determine whether you have bought a powerful new addition to your system, or a nice conversation piece that soaks up power and wastes a valuable slot in your computer.

Three Categories of Co-processors

The various microprocessor cards available for the Apple come in three major categories: devices which open up whole new worlds of high performance commercial software, microprocessors which enhance the performance of programs written in Applesoft, Pascal or FORTRAN, and devices which should only be purchased by hardware hackers and machine language enthusiasts.

Enhancing Commercial Software or Getting IBM PC Compatibility

For users of commercial software, the choices are fairly strictly limited to the high speed 6502C from Titan Technologies (the Accelerator), some version of the standard Z-80A CP/M system from Microsoft (the SoftCard), or the high performance Z-80B CP/M system from PCPI (the Appli-Card). These three cards and some other similar ones from other manufacturers as well as related //c expansion modules each can provide an enormous enhancement to the power and versatility of an Apple. The unique strengths of each of these will be discussed in detail later in the Chapter.

Another related area of concern for many Apple owners is the question of achieving some level of compatibility with the IBM PC. A great deal of the software that runs on the IBM PC is available to run on the Apple's 6502 or on a Z-80, and most of those programs actually run faster on an Apple Accelerator or a PCPI Z-80B than they do in the IBM PC. However, there are integrated (Lotus 1-2-3), multitasking (Concurrent CP/M 86) and windowing (Visi-On, Microsoft Windows) applications which have to have a large amount of memory and are thus designed only for the 8088. Another reason for wanting to get some measure of compatibility arises for people who would like to exchange work among various machines.

For 6502 based software on the Apple there is a very nice solution: purchase a Quadlink board for the IBM PC. The Quadlink is essentially a complete 40 column Apple II+ on a card along with a system of altering the behavior of the PC's disk drives. This permits the PC to use the Apple's disks in full Apple emulation mode.

Running MS DOS and CP/M 86 software on the Apple is easy in theory since four companies make 8088 based expansion boards. However, there is no way to make an Apple Disk II drive read IBM PC diskettes. Coprocessor boards with 8088 microprocessors may be interesting for hackers and can be useful for programmers who work in BASIC or Pascal, but they're not a good choice for folks whose principal interest is to run applications software.

However, a really excellent option for someone considering buying an IBM PC is to get the Rana 8086/2 instead (see Chapter 30). This is effectively a complete IBM PC including two 360K disk drives, 256K of RAM (expandable to 512K), and a color graphics system. It is interfaced to the Apple bus so the drives can also be used for DOS 3.3 and CP/M. Since it is based on the 8086 rather than the 8088, and since it has more high resolution display modes than an IBM PC, it is a very interesting option, particularly in the light of decisions by Microsoft, Lotus and Apple to directly support the product.

Enhancing Math Processing and High Level Languages

Another set of reasons for purchasing a coprocessor board is to improve the programming environment of the Apple. There are very few languages which just aren't available for the 6502. However, for the most part the point is to enhance the performance of existing programs written in Applesoft BASIC, FORTRAN, or Apple Pascal.

The two major areas of enhancement are; the speed with which mathematical calculations are performed, and the size of the memory range available for programs. It is possible to improve on the memory range simply by adding RAM and doing bank switching (see Chapters 25 and 40), but several new language interpreters and compilers are now available for 68000 boards for the Apple which offer huge increases of "linear program space."

Numerical calculations are not extremely important in many programs; however, in a wide variety of scientific, engineering, graphics, and business programs, math is a severe bottleneck. There are several coprocessor cards which let you run programs you have already written in Applesoft BASIC or Pascal, but which are able to intervene whenever a calculation needs to be done.

You can get a modest improvement in math processing speed by using the ALF 8088 card and a more substantial improvement with a 68000 board from Saybrook or ETC. These two processors still require about 40 and 20 microseconds respectively for a 16 bit multiplication which takes 500 microseconds with the 6502, and more complex manipulations must be built

up step-by-step. There are also pure math chips such as the 66016 from Synertek which can do that 16 bit multiplication in 0.1 microseconds, but which can't do much else without a lot of software and hardware support.

The ALF card can also be equipped with a "floating point processor" called the 8087, and this provides a hundredfold increase in calculation speed over and above the 8088's performance for a broad variety of functions such as exponentiation and trigonometric functions. Yet another option is to get a board which does high speed vector math for graphics with the NEC 7220 (see Chapter 7).

Cards for Machine Language Programmers

Some Z-80 and 8088 cards aren't of much use for applications or high level language work, simply as a result of poor systems software or unsuccessful card design by manufacturers. There are, however, a couple of very interesting devices which are offered for sale by companies which just happen to be interested in high performance microprocessors for their own sake (i.e., 68000 DTACK Grande). These are new products which do not yet have a sufficient software base to interest applications oriented users. Other than these, of course, any coprocessor card is fair game with the caveat that if you cannot get good documentation (hopefully source code) on the idiosyncrasies of a particular system (I/O services, Apple interface, etc.), you may end up in a very frustrating quagmire.

A Survey of Microprocessors

The very first eight bit microprocessor was the Intel 8008, which was completed in January of 1972. It's worthwhile to recall that when Neil Armstrong put his spacesuited boot down in the dust of the Lunar Sea of Tranquility in 1969, although there had already been considerable advances in small but powerful computational devices, no one had even drawn a first sketch for a microprocessor. And for the most part electronic component manufacturers couldn't see why anyone would want to build one.

Nonetheless, Fred Hoff and Frederico Faggin at Intel were interested in the technological problem of building such a device, and when, in 1969, two companies, Busicom and Datapoint, contacted Intel about the possibility of building a single chip CPU, the wheels were set in motion. Busicom was interested in a four bit device, but Datapoint was hoping Intel could come up with an eight bit CPU to handle ASCII codes and communication for a CRT terminal. In existing terminals, the functions of the CPU were built up from 10 or 15 chips, and a single chip CPU could provide a sharp drop in price of the electronics for the terminal.

The Intel 4004 built for Busicom (finished in November of 1971) gradually disappeared from use, but the 8008 was an immediate failure that Datapoint refused to buy once it was finished. However, the design choices made in creating the 8008 have heavily influenced nearly all subsequent microprocessors. The reason the 8008 wasn't successful had to do with the problems of creating very small transistors which could work at high speed. It processed perfectly well, but it did its work about 10 times more slowly than the older multi-chip design. The fundamental architecture of the device was retained and enhanced in two descendant microprocessors, the Intel 8080, and the Motorola 6800, both completed in 1973.

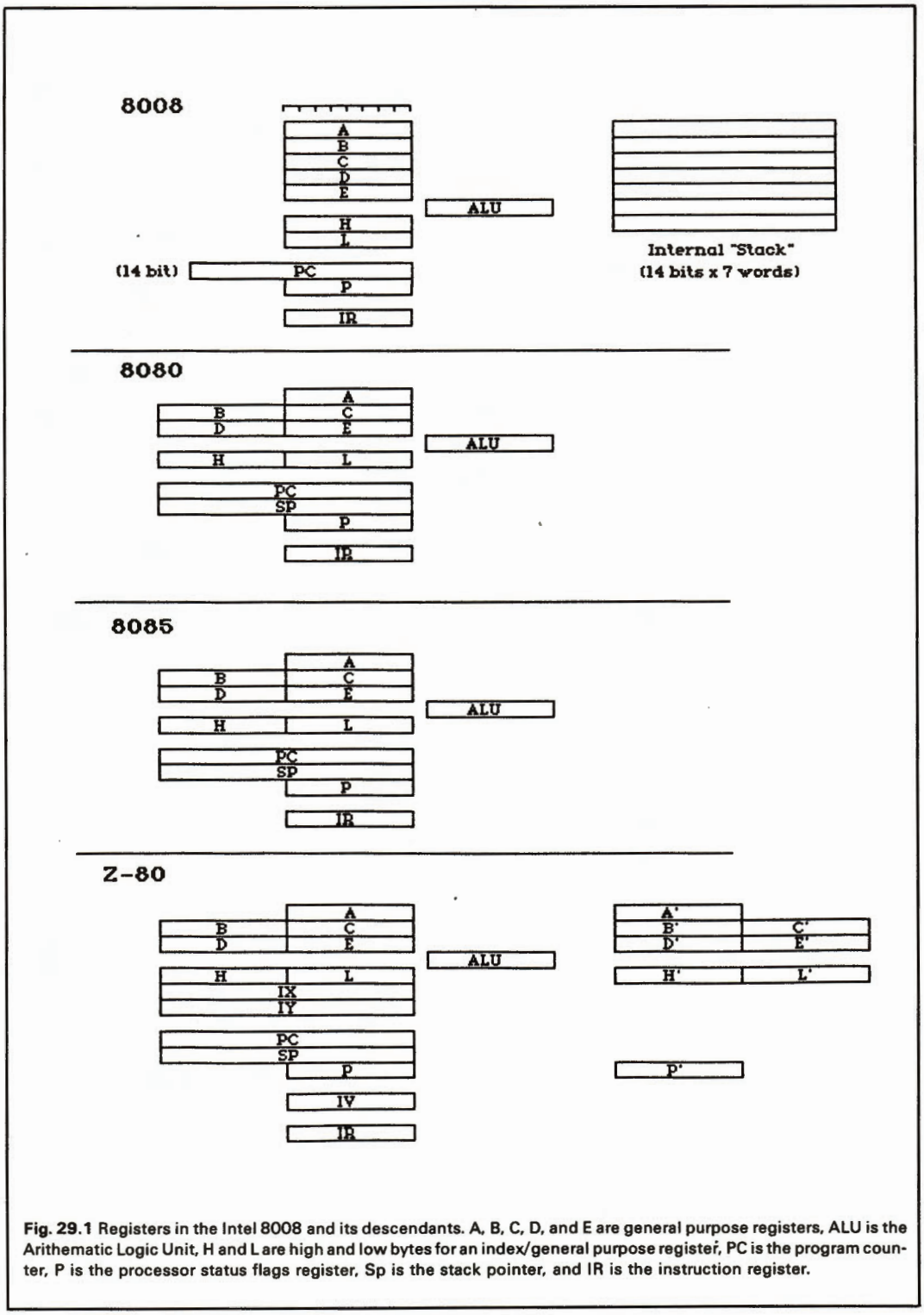


Fig. 29.1 Registers in the Intel 8008 and its descendants. A, B, C, D, and E are general purpose registers, ALU is the Arithematic Logic Unit, H and L are high and low bytes for an index/general purpose register, PC is the program counter, P is the processor status flags register, Sp is the stack pointer, and IR is the instruction register.

A new manufacturing technology called NMOS (see Chapter 13) solved the speed problem, but the two companies followed different routes in enhancing the architecture. These two routes are reflected in descendants of the 8080 such as the 8088 (in the IBM PC), and the Apple's 6502 and the Lisa's 68000.

Design Elements of the 8008, 8080 and 6800

The original 8008 design emphasized on-chip storage of a substantial amount of data and address information. There was an eight bit accumulator (A), four general purpose eight bit data registers (B, C, D, E), and two eight bit registers used for constructing data addresses, an effective "data counter" (H and L; see Figure 29.1). In addition, there were seven 14 bit registers which made up the "stack" of the 8008. An eighth 14 bit register served as the program counter (see Chapter 27) which could address 16,384 locations in external memory.

The layout of the programmable registers of the 8080 and of the 6800 are somewhat different, both from each other and from the 8008 (see Figure 29.5). In the 8080, all of the data and address registers have been maintained, except that they can be addressed in pairs as 16 bit registers. The standard programmer's model for the 8080 shows an accumulator A, data registers BC and DE, and an address register HL. The stack has been banished from the microprocessor. In its place is a new 16 bit stack pointer which can place the stack (see Chaper 27) anywhere in external memory.

The program counter has been expanded to 16 bits so that a full 65,536 (64K) locations can be addressed. The instruction set of the 8080 emphasizes the construction of full addresses by use of the onboard data registers, placing the address in HL, and then using the address in HL as the "implied" extension (see Chapter 28) for essentially all data moves.

The 6800 has banished both the stack and the data registers to external memory. Just two eight bit accumulators are retained. The H and L registers are again combined for potential use as a 16 bit data counter, but the 6800's instructions tend toward more complex use of this register set for "indexing" of addresses (see Chapter 28). Like the 8080, there is now a full 16 bit program counter, and a 16 bit stack pointer.

The 6800 design differs from the 8080 primarily in that the 6800 must fetch bytes from memory much more frequently than the 8080. The advantage is far greater flexibility, and the disadvantage might be that each step takes longer. However, the 6800 design executes the various steps of each instruction more rapidly than the 8080, so the 6800 is, in a sense, able to afford this luxury. The TMS 9900 from Texas Instruments carries this sort of thing even farther in that there is a 16 bit program counter, and a 16 bit pointer called the workspace register, and that's about it. Everything else is done out in memory. This TMS design and the 6800 both reflect minicomputer design in which special registers near the CPU are not particularly important.

The Third Generation

The great flourishing of microcomputers, and the creation of most microcomputer software, was done in the environment provided by a third generation of microprocessors which came to market between 1975 and 1978. The Z-80 and the 6502 were both released during 1975,

and therefore were the two chips which got designed into most of the microcomputers which first appeared in 1976 and 1977, including the Apple.

It really wasn't obvious until 1977 that microcomputers could be important commercially. But by 1978 it was very apparent, and a fairly large number of new microprocessors and enhanced microprocessors began appearing. However, the large software base of the 6502 and the Z-80 placed major constraints on the market. There was no reason to use a slightly improved microprocessor if it would not run existing software.

The Z-80 and the 8085

The Intel 8085 (from 1978) is essentially identical to the original 8080 from the programmers point of view. It mostly represents an improved package in terms of the inclusion on chip of many of the support circuits needed for the 8080, and the addition of a more elaborate system for handling interrupts. This chip is in many ways a catch-up product which implements many of the changes that Zilog had built into its Z-80 (from 1975) update of the 8080.

The Z-80 kept the basic register structure of the 8080, but added a variety of features, mostly having to do with interrupts. The most striking difference in the programming model in Figure 29.1 is that the A, BC, DE, HL, and status registers are duplicated as A', BC', DE', HL', and P' (pronounced A prime, etc.) During an interrupt, the complete contents of the program registers can be transferred into the primes. There are also three completely new registers. One is an eight bit interrupt vector, but the other two are 16 bit index registers. These permit much more elaborate address construction. Finally, Zilog added some circuitry which could supervise the refresh of dynamic RAMs (See Chapter 21).

With all of these new hardware resources, the instruction set of the Z-80 is greatly expanded over the 8080 and 8085. The 8080 has about 80 instructions, but the Z-80 has more than 220 instructions. The CP/M operating system had been written to run on the 8080, so the Z-80A was made upwardly compatible, meaning that machine language written for the 8080 will run on the Z-80. Many programmers therefore chose to write all their software in 8080 code so that it could work on both 8080 and Z-80 systems, and, as a result, most of the Z-80's extra instructions and addressing modes are rarely used.

Speeding Up the Z-80

Enhancements of the Z-80 have focused almost entirely on speed of operation, since its registers and instruction set are considered to be more than sufficient for most tasks. The original Z-80A has been available in a two megahertz and a four megahertz version. The Z-80B can operate with a six megahertz clock and the Z-80H can handle eight megahertz.

These enhancements are done without changing anything in the circuit layout or "photolithographic mask" of the chip; rather, they reflect technological advances in chip fabrication technology. The principal drag on speed is "stray capacitance" between circuit parts (see Chapter 13). Capacitance can be reduced by physically reducing the size of the "plates," so speed is increased by shrinking the microprocessor. The principal limit here is that resistances also get reduced, so current flow goes up and that means more heat in a small space.

The Z-80 has been popular in designs because of its improved electrical interface and because it includes many support features which required extra chips in 8080 computers. Its greater programming versatility has had comparatively little impact in the microcomputer world

because of the near universal use of 8080 machine language even in Z-80 systems. Intel noted all this, and when they released their own update of the 8080 in 1978 (the 8085), they omitted all the programming enhancements and stuck strictly to the electrical and support improvements.

Z-80 Coprocessors for the Apple Bus

The SoftCard from Microsoft and the Appli-Card from PCPI (Personal Computer Products Inc.) are both very popular Z-80 coprocessor cards for the Apple, but they operate in two very different ways, and, with a few exceptions, cannot run the same software. The Appli-Card with a six MHz Z-80B is the best choice if your primary objective is high performance with WordStar and dBase II. None of the Microsoft products can match the Appli-Card's speed and flexibility for those two programs.

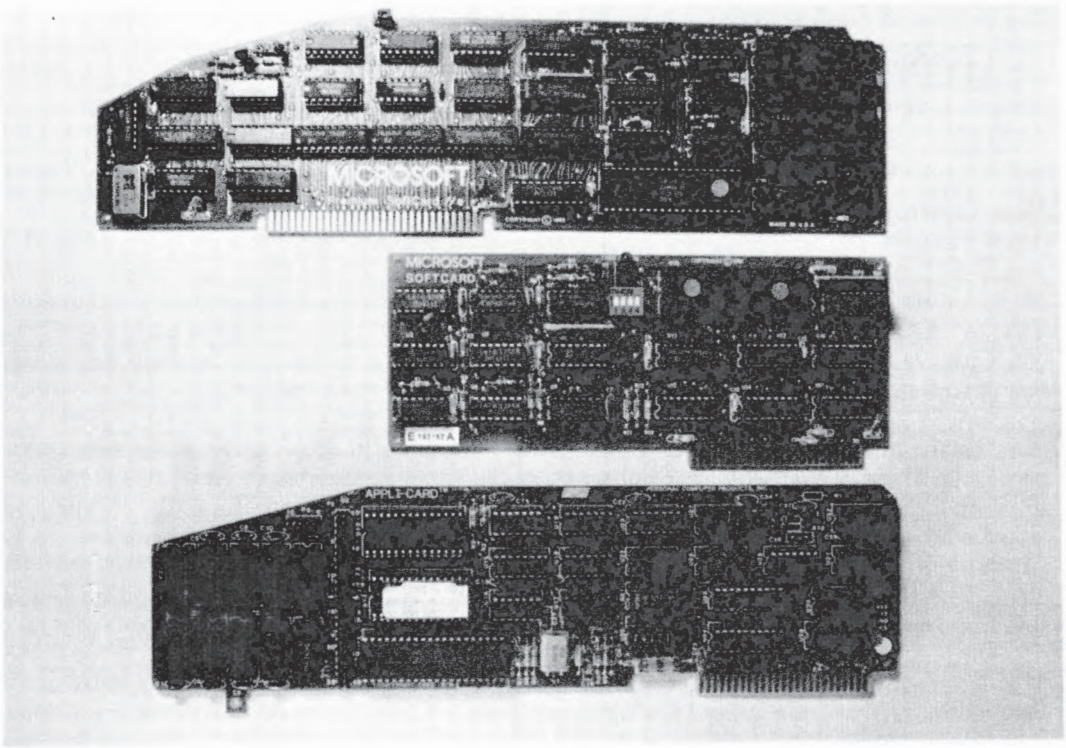


Fig. 29.2 Microsoft Premium SoftCard //e (top), MicroSoft SoftCard (middle) and PCPI Appli-Card (bottom).

For almost any other CP/M application, you're far better off with the SoftCard. If you own a //e, you need to get the Softcard //e and not the Premium Softcard //e if you want to reap the full advantages of going with Microsoft. The Z-Card from ALS, and the Applied Engineering Z-80 card, both use the Z- 80A and will usually run SoftCard style software. There are also two MHz and four MHz versions of the Appli-Card which use the Z-80A, but they don't really

give you enough performance to justify the other sacrifices involved in not getting a SoftCard. The CP/M Card from ALS, the Gold Card from Digital Research and the Premium SoftCard //e from Microsoft all use a six MHz Z-80B and can require special versions of programs. However, because of various design considerations, they don't match the performance of the six MHz Appli-Card.

The Microsoft SoftCard

The SoftCard is the first and foremost coprocessor for the Apple. It has a two MHz Z-80A and gives you near total access to the complete world of CP/M based software. The extra bonus for programmers is that it is manufactured by Microsoft, so you are assured compatibility with Microsoft's MBASIC, COBOL, FORTRAN and LISP, as well as such languages as ADA and PL-I from other vendors. Microsoft has written modified versions of their own languages for the Apple, and these versions usually will not run on Apple Z-80 cards from other manufacturers.

The documentation you get with any Microsoft product is thorough and nicely packaged. This was remarkable in 1980, and in 1984 it is still exemplary. Almost anything you might need to know comes packaged with the product. With Microsoft, though, you are out of luck if there is something you need to know which isn't in the documentation. At some companies, no one knows the answers to very technical questions. At Microsoft, they know but they won't tell you.

Bus Timing

Since the SoftCard was designed in 1980 when 64K of RAM could cost as much as $500, it comes with no RAM of its own on board. Rather, it turns off the 6502 and siezes control of the motherboard RAM you already own. One distinct advantage of this approach is that along with the RAM, it gets direct access to all of the Apple's I/O port addresses and softswitches (see Chapter 26). This has made it convenient for programs running on the SoftCard to take full advantage of the Apple's built-in peripheral slots and video system.

Sharing the motherboard RAM is therefore a nice idea, but it does present some very thorny problems. Those of you who are familiar with the Apple video system (see Chapter 25) will know that during half of every microsecond, the Video Display Generator (VDG) is supposed to get control of the RAM. The 6502 and the VDG have a very nice system worked out for passing control back and forth during each microsecond, all based around the 6502's one MHz clock. Into this delicately balanced situation the engineers at Microsoft (chiefly Don Burtis) had to insert a two MHz Z-80A.

This feat requires that the Z-80 be phase locked to the Apple's system clock and video needs, yet generate its own customized timing signal. In effect, the Z-80 must slow down to the 6502's speed for half of every microsecond, but then it must zip through two very fast clock pulses in the other half. The result is a rather odd synchopated clock rhythm which works as follows.

The Z-80 watches the seven MHz clock signal available on the Apple bus and it also watches the one MHz system clock. During the VDG's half of the microsecond, the SoftCard divides the seven MHz signal into a 3.5 MHz signal. It does one full cycle at this speed, and gets a little more than halfway into a second cycle at 3.5 MHz, but then, during the next half of the microsecond, it slows down to what amounts to the 6502's one MHz speed. It does its data moves at this time. Thus, the average rate at which the Z-80 proceeds comes out a little bit over two MHz. It slows down to 6502 speed for half of every microsecond so it can follow normal conventions for sharing RAM with the Apple's VDG.

This is all so tricky, and so precise, that the slight timing changes on the //e relative to the II/II+ can render a SoftCard unsafe at any speed. As a result, Microsoft released the SoftCard //e (not to be confused with the Premium SoftCard //e) which works fine in the //e. The only differences between a SoftCard and a Softcard //e are a couple of tiny little trace changes between a few of the chips, so the two cards actually look identical to the untrained eye. This little glitch came as a surprise to an unbelieving Microsoft as hundreds of SoftCards were returned as defective within the first month after the release of the //e.

DMA and RDY

The way that the Z-80 turns off the 6502 is by use of the Apple's DMA circuitry (see Chapter 27). This permits it to disconnect the 6502 from the data and address bus and also freezes the 6502 in its tracks by disconnecting the system clock from the 6502. One potential problem with leaving the clock disconnected is that the 6502 is a "dynamic" microprocessor which means that it has to get an incoming clock pulse at least once every 40 microseconds or its registers go haywire. Therefore, the SoftCard must let the 6502 do something every now and then in order to help keep it out of trouble.

The way this is done is to let up on the DMA line periodically. However, in order to prevent the 6502 from actually doing something on the bus, the SoftCard always pulls the 6502's RDY line (see Chapter 27) while the clock is reconnected. This lets the 6502 refresh itself without causing any trouble elsewhere. This sequence takes place every four or five microseconds, but it is supervised in such a way that it requires no special time or attention from the Z-80. This is because, in a final clever flourish, the Z-80's special dynamic RAM refresh circuitry was put in charge of refreshing the 6502.

The Z-80 refresh circuitry operates only while the Z-80 is busy decoding an instruction and so never gets in the way while the Z-80 needs to use the address and data bus. Thus the Z-80 never has to slow down its processing to take care of refreshing the 6502. The Apple's VDG takes care of refreshing the motherboard RAM chips, so the Z-80 refresh circuitry would otherwise be wasted.

Exchanging Control Between SoftCard and the 6502

As the SoftCard steps its way through a program, it sometimes needs special help from the 6502. This is in part because most Apple peripheral cards have built-in I/O routines written in 6502 machine language. To use these cards, the SoftCard can store a description of its needs in some agreed upon memory location and then turn control of the Apple back over to the 6502. The 6502 looks at the request, runs through any necessary 6502 machine language routines, and then leaves the data in an agreed upon place where the Z-80 can find it.

Thus, both the 6502 and the Z-80 can be involved in the execution of some larger task, but since they share a single set of RAM, address bus lines and data lines, they can't both be active at the same time. The way that control is passed back and forth is to ensure that, at all times, either the DMA line is halting the 6502, or the Z-80 "WAIT" line is halting the SoftCard.

The exchange is based on the I/0 Select line (see Chapter 22) on the accessory slot. If the Z-80 is, for instance, in slot 4, then any time the 6502 issues an address between $C400 and $C4FF the I/O Select line to slot 4 will turn on. This toggles a flip-flop on the SoftCard which both turns off the Z-80 WAIT signal and turns on the Apple's DMA circuitry. The Z-80 can then turn itself off by doing another write to this location ($E400 to $E4FF after translation—see below). This write toggles the flip-flop the other way, thus throwing the Z-80 back into its WAIT state and releasing the 6502 from the control of the DMA circuitry.

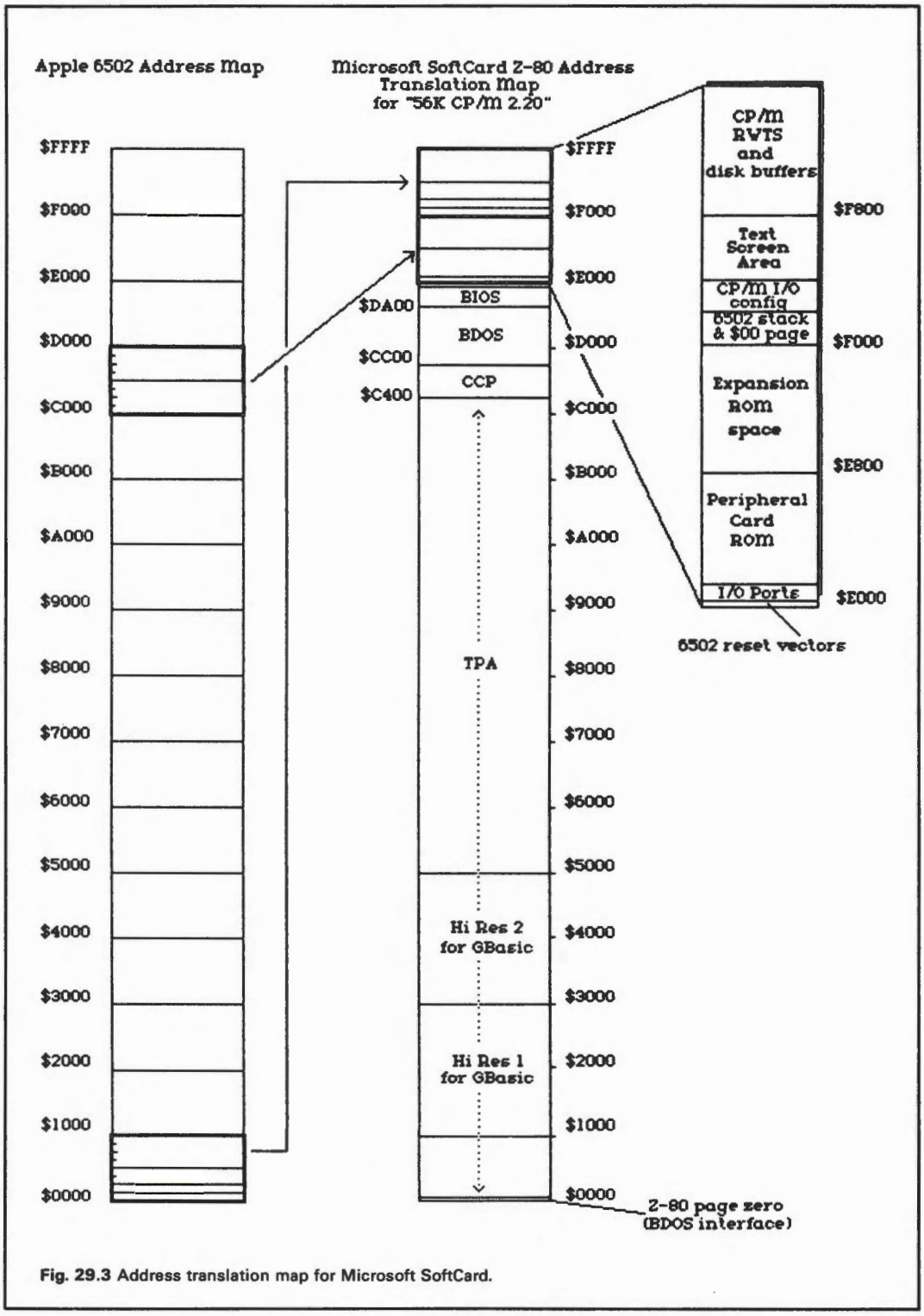


Fig. 29.3 Address translation map for Microsoft SoftCard.

Rearranging the Address Space

The RAM and I/O locations in the Apple are arranged in such a way as to adjust to the special needs of the 6502 and the Apple's ROM based Monitor and Applesoft Interpreter (see Chapter 21, Figure 21.4). However, most of the details of this arrangement are just so much excess baggage for the Z-80. In order to get a full, wide open address space which is arranged for the Z-80 and for CP/M, the SoftCard uses a trick called "address translation" to effectively rearrange the locations on the Apple motherboard. All that is needed to carry this out is the interception and modification of some outgoing address lines on the SoftCard. Nothing on the Apple's motherboard is actually moved, the Z-80 just sees them in a different place than they really are.

The results of the address translation include moving the "zero page" (see Chapter 21) all the way up to the $F0 page, putting the $C000 range of I/O locations (see Chapter 22) up at $E000, and moving the whole ROM range of $D000 through $FFFF down to $B000 through $DFFF. These three shuffles have the result of moving all of the special locations up to the top of the Z-80's address space so everything below is clear. With CP/M in place, there is thus 44K of free linear RAM addresses.

When you add a 16K card, older versions of Microsoft CP/M were unable to do the D/D* bank switching (see Chapter 25), so you only got to use an additional 12K for a total of 56K of RAM (see Figure 29.3). The 4K of RAM which was unused became a popular place for other manufacturers to put their special driver routines for hard disks or high density floppies. However, Microsoft later upgraded their CP/M to use this last 4K. As a result, you got a full 60K of RAM to use, but a lot of the drivers from other companies wouldn't work any more. You must therefore attend to the need for "56K CP/M" or "60K CP/M" when you buy a non-standard disk drive system to use with a SoftCard.

The Z-80B and the PCPI Appli-Card

From Microsoft's point of view, the Appli-Card is the big fish that got away. The folks at PCPI spotted this device for what it was worth and snatched it up from its designers just as Microsoft was getting interested. Of the four Z-80B boards available for the Apple, this one is the hands down champion for several hardware, software and packaging reasons. The six MHz Z-80B on the Microsoft Premium Softcard //e squanders its speed on the slow //e video system it provides and on a slow interface with the 6502, so you get little noticeable enhancement of the sluggish performance of WordStar. Further, neither the Premium Softcard //e nor the ALS Z-80B CP/M Card have any provision for direct access to more than 64K of RAM, so you can't get very high speed sorts with dBase II (see Chapter 24) or take full advantage of the banked version of CP/M 3.0 (see Chapters 35 and 36).

Both the Gold Card from Digital Research and the Appli-Card from PCPI have "on-card" expansion buses for adding another 128K of fast RAM (see Figure 29.4). Hardware performance is roughly similar in these two cards, but the system software "philosophy" is different. The Appli-Card offers a highly optimized CP/M version 2.0 which uses extra RAM as a 128K RAM disk. It also has available support software for use with dozens of peripherals from other manufacturers. The Gold Card is configured to run CP/M 3.0 and to use the extra 128K of RAM as "sector" and "directory" buffers (see Chapters 32 and 36). The PCPI RAM disk approach is better if you can work in the confines of an effective 128K "disk drive." If, instead, you want to improve access speed to very large files on a hard disk, then you may be better off with CP/M 3.0

All four of the Z-80B cards have 64K of their own RAM on the board with them (see Figure 29.2). They are thus really complete microcomputers on a card, and deal with the Apple as if it were a separate computer. This lets them get rapid access to programs and data in their own memory without having to contend with the timing intricacies of using the motherboard RAM (see above). However, this also means that they lose direct access to the Apple's I/O ports and softswitches and thus become incompatible with some software written for the shared RAM design of the older SoftCard. Therefore, buying the Microsoft Premium Softcard //e does not ensure software compatibility with the Microsoft SoftCard //e.

The Z-80B cards are full scale coprocessors in that the 6502 continues to run simultaneously while they are working. The Apple's ports, RAM and 6502 serve as a supplementary I/O system for the Z-80B. Whenever the Z-80B wants information from the keyboard or data from the disk drive, it makes a request to the 6502. The 6502 then goes about reading the keyboard or collecting a sector of data from the disk drive. The information is first loaded into the Apple's own RAM, and then it is handed over to the Z-80B. The Appli-Card and the CP/M Card exchange data with the 6502 as if the two were in separate computers connected by a parallel port. This is the same "two computers and a port" concept used in Z-80 systems for the //c.

The Premium SoftCard //e goes in the auxiliary slot, so the Z-80B/6502 interface is a little trickier. The auxiliary slot provides no access to the Apple's DMA, IRQ or RDY lines. Further, although the "row address" lines (see Chapter 21) connect to the auxiliary slot, there is no connection to the actual address bus. The burden therefore falls upon the 6502 to look up into the Z-80 address space to see when it is needed. To execute an I/O request, the 6502 turns off the Z-80, moves the data into or out of the Z-80's address space in auxiliary RAM, and then turns the Z-80 back on again and returns to its waiting loop.

Error Handling and Speed

The Appli-Card comes with a very powerful hardware design and a version of CP/M which is much kinder to the user than the versions distributed by Microsoft. If you use a SoftCard and have gotten a BDOS Error after several hours of writing in Wordstar, you know the horror of the sudden and unwelcome appearance of the system prompt which confirms your fears that all is lost. If there is a BDOS Error while using WordStar on the Appli-Card, it's no problem. You come back up into WordStar and can take evasive action with the loss of no more than three or four words of text.

The crucial advantage of the Appli-Card hardware is the inclusion of a separate expansion bus on the card. You can use this feature to add another 256K of RAM to the Appli-Card. Not only are PCPI RAM expansions comparatively inexpensive, but they operate five to 10 times faster than RAM extensions used with the Microsoft or ALS Z-80B cards. When the Premium SoftCard //e wants to get access to additional RAM, it has to inform the 6502, get turned off, wait while the 6502 does the data move at one MHz, and only then regain control. The Appli-Card uses all of its RAM at a full six MHz. This is why a large dBase II sort done on a RAM disk can take six or seven minutes with the Premium Softcard //e, but only 90 seconds with the Appli-Card.

The design of the Appli-Card is so well suited to running WordStar that MicroPro decided to buy a huge quantity of Appli-Cards to give away free to purchasers of WordStar and even wrote a new enhanced version, WordStar 3.3, specifically to run on the Appli-Card (which they market as the StarCard). The big difference between WordStar on the Appli-Card and WordStar on a Microsoft Z-80 is the overwhelming feeling of spontaneity as you work.

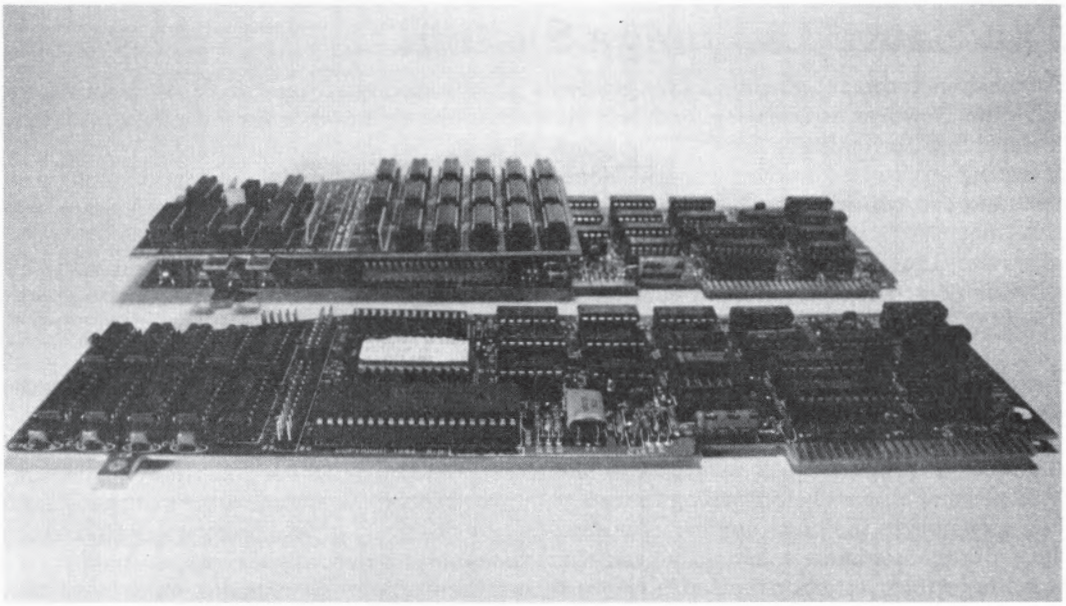


Fig. 29.4 PCPI Appli-Card with (top) and without (bottom) a 128K RAM extender. There is 64K of RAM on the card itself, but the piggy back board provides an additional 128K for use as a very high speed RAM disk.

Because the SoftCard runs at two MHz, it cannot keep up with most touch typists in a computationally intensive word processing program such as WordStar. The Premium SoftCard //e does the computations fast enough, but it forces you to use the //e 80 column card. And in that system the relatively slow scrolling and screen updating deprive you of much of the advantage provided by the high speed Z-80. With a six MHz Appli-Card and the Ultraterm (//e), or enhanced ALS Smarterm (see Chapter 6) for 80 column display, you can type at over 60 words per minute. If you already own an external CRT terminal, such as a Televideo, you can connect it directly to a serial port on the Appli-Card and get similar speed effects.

Software Considerations

The Appli-Card also takes advantage of its complete coprocessor relationship with the 6502 to offer two additional major software features. For those looking to minimize expense, you can run the Appli-Card without an 80 column card since it will generate a 70 column display from software using the Apple's standard high res graphics screen. If, on the other hand, your tendency is towards adding high powered enhancements such as hard disks and external RAM devices, the Appli-Card software is designed for very easy integration of new equipment.

Although the Apple versions of many CP/M applications programs will work only on the SoftCard, you can usually get versions for the Appli-Card. Most of these programs have been available in a standardized version for S-100 computers with eight inch disk drives, and a SoftCard version for the Apple. The Appli-Card will run the standardized version but you need to buy the software from a vendor who will move it onto Apple format 5 1/4 inch disks (which they sometimes call Microsoft format). A large number of Appli-Cards were sold through the MicroPro/WordStar arrangement and as part of the Franklin 1200 ACE computer, so software vendors are increasingly willing to provide Appli-Card versions.

The Savvy Language System

For most microcomputer users, a Z-80 processor is useful because it can run CP/M applications software. However, it is a very flexible 8 bit microprocessor in other regards and Jim Dowe of Excalibur Technologies chose to use a Z-80 as a basis for designing a completely different operating environment called "Savvy." None of the popular commercial applications software packages can run in this environment, so Excalibur has developed a complete series of accounting, word processing, database, and spreadsheet software for Savvy. These all work adequately, but it's clear that they are the product of a single, small, hard pressed group of programmers. Thus this is not a package which will be useful for the great majority of applications oriented computer users.

The reason for all this redesign effort at Excalibur is Dowe's interest in inserting a "pattern recognition" level between the words the user types at the keyboard and the commands executed by the program. Before the dawn of the great age of microcomputers, you always addressed a computer with very precise and literal commands which you either looked up from racks of binders before typing them into punch cards or memorized after months of use. To accommodate the huge number of new users, the designers of microcomputers shifted to shorter lists of commands, then to help screens which reminded you of your options, and finally to menus which let you point out a choice rather than type in a command. More recently, Apple has shifted away from verbal commands towards Lisa's pictorial icons for presenting choices.

Savvy still requires you to type commands into the machine, but it lets you be imprecise. The flexibility for the user comes at two levels. First, Savvy can often recognize a word even if it is misspelled or preceded by extraneous words: "I want you to ron the program" can get spotted as the command RUN. Secondly, it is convenient to program Savvy with synonyms for a command so that "go," "start," "begin," and "do something" can all be interpreted as RUN.

To make the product useful, you have to be interested in programming in the Savvy language. The applications packages provided by Excalibur Technologies are intended to be modified by a Savvy programmer in such a way as to become specialized for a given application. You can, for instance, use the synonym "associate" feature to modify their inventory control program so that your employees can type in "How many pliers did we sell?" rather than a generalized command. Thus the best environment for Savvy is one in which a programmer interested in gaining skill in the language is creating applications for casual users who will not have extensive training in using the software.

Robot Control Language for RB5X

For the most part, although computer users like things to be easy to use, they place enormous value on getting the computer to do exactly what they want it to do, exactly when they want it to be done. Ambiguity and imprecision are generally considered suspect. This is one of the serious drawbacks with most current voice recognition systems for computers (see Chapter 11). Those systems use pattern recognition to match up the sound you uttered with the stored and compressed digital images of known word sounds.

If the recognition requirement is too strenuous, the computer will not understand you, coming back with a report of no match found. However, if the requirements are too relaxed it may make an incorrect pairing and issue the wrong command. The effectiveness of the system is limited by the amount of available memory for storing detailed patterns and the processing power available for sorting through them.

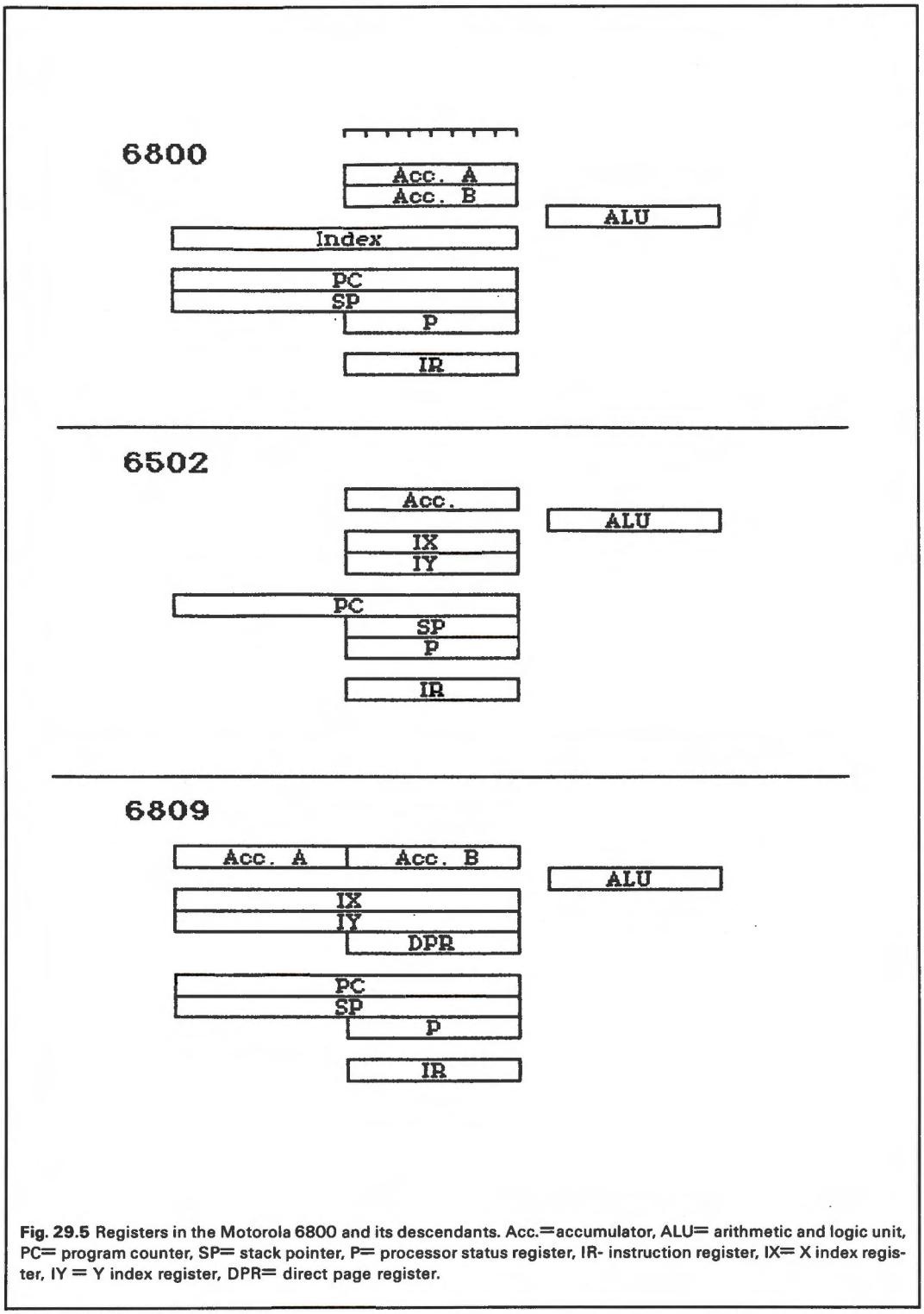


Fig. 29.5 Registers in the Motorola 6800 and its descendants. Acc.=accumulator, ALU= arithmetic and logic unit, PC= program counter, SP= stack pointer, P= processor status register, IR- instruction register, IX= X index register, IY = Y index register, DPR= direct page register.

One advantage of Savvy in this regard is that it has a virtual memory management scheme for using far more than 64K of RAM all of which it can use to store very detailed images of a large number of known commands. This can make command recognition much more secure and versatile than with most current voice recognition systems. Therefore, the RB Robot Corporation decided to use Savvy as the basis for a control language for their RB5X robot. You have to walk over to a computer with some confidence about what you're up to, but a robot can walk up to someone who has no reason to know how to operate or command it. In this situation, the associative power of Savvy becomes more a necessity than a convenience.

The 6502 and the 6809

The 6502 from MOS Technology emphasizes the basic commitment of the Motorola 6800 in that it relies on fast and efficient instruction execution and comparatively heavy use of external memory. One important trick that greatly enhances the 6502 relative to the 6800 is the special use of the zero page and the $01 page of external memory.

All of the addresses in the zero page are effectively eight bit addresses (i.e., $0023, $00EF, etc.) which can be expressed with a single address byte, so it is much faster and simpler to access than addresses in the remaining 255 memory pages with 16 bit addresses (see Chapter 21, Figure 21.1 and Figure 21.4). The 6502 therefore uses the zero page as if it were a 256 byte data and address register system. Similarly, the $01 page is used as a 256 byte stack. All stack addresses are treated as eight bit addresses, but $01 is added in the high byte just before the address is announced.

Although the 6502 machine language includes 151 instructions and uses some fairly complex addressing modes, none of its instructions take longer than seven clock cycles, while similar instruction on the Z-80 may require 20 clock cycles. With a one megahertz clock rate, no 6502 instruction takes longer than seven microseconds, while a Z-80 with a two megahertz clock may require 10 microseconds for the same instruction. This is why the performance of a one megahertz 6502 is directly comparable with the standard two megahertz Z-80 and 8085 systems. In many situations, although the 6502 has a clock speed which is twice as slow, the rate at which it processes instructions can be twice as fast.

The two eight bit index registers on the 6502 are also quite important. Although the Z-80 has two 16 bit index registers, they are not usually used because the 8080 doesn't have any index registers and most Z-80 programs are written in 8080 machine language (see above). As a result, the 6502 is capable of very elaborate and expressive addressing as well as high speed even though its register set looks quite simple on a first glance (see Figure 29.5).

Some Other 6502 Based Computers

The fact that you are reading this book nine years after the 6502 was designed attests to its popularity among programmers and designers. The 6502 is used in the PET, the Atari, the Commodore, and the Ohio Scientific microcomputers among others, all of which have no software compatibility with the Apple II. Some companies, such as Franklin and Basis, have designed Apple II-like products around the 6502, but you cannot count on these machines to run system software from Apple such as the ProDOS operating system.

Apple Computer has licensed Applesoft, the Monitor, and DOS 3.3 to Central Point Software for use by Quadram on their Quadlink 6502 board for the IBM PC (see Figure 29.6). The

Quadlink takes over the PC's disk drives, gives you access to the modem and printer cards in the PC, and it also lets you configure PC memory cards as Apple RAM disks, but it does not provide any means for getting an 80 column text screen display. Since it has no slots of its own, you cannot enhance it with any Apple plug-in products.

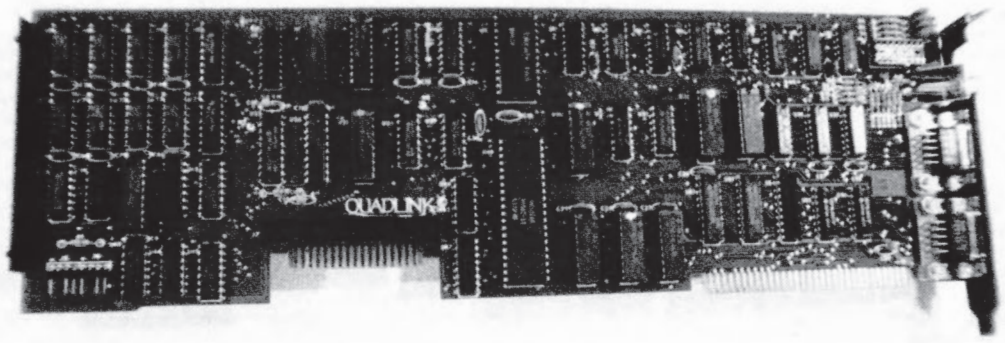


Fig. 29.6 Quadlink board for the IBM PC.

Higher Speed with the 6502C

Because of the enormous software base of the 6502, the major efforts at enhancement have had to do with providing increased speed, much like the situation with the Z-80. The 6502B used in the Apple //e and Apple III can run at two megahertz, and the 6502C used in the Accelerator Card from Titan Technologies runs at four megahertz. Most of the Apple II video hardware and some of the system software require that the 6502 on the motherboard run at one MHz. However, you can add a plug-in card which will run your applications software and your own programs on a high speed 6502.

The Accelerator is a complete "Apple on a card" which has a full complement of 64K of RAM (see Figure 29.7). It generally keeps the Apple's 6502 turned off by means of the DMA line; however, it must synchronize with the Apple clock in order to load data into the video display areas in the motherboard RAM (see Chapter 7). Its greatest power is for speeding up computation intensive programs such as statistics and spreadsheets. It can also speed up plotting in the high res screen since the plotting routines can be a major speed bottleneck (see Chapter 41). The speed increases apply for Applesoft BASIC and Pascal, but don't help much for programs which are slowed down by heavy I/O tasks such as sorting on the disk drive.

The board was actually designed by the same folks at Number Nine Computer Engineering who designed the NNGS ultra high resolution graphics board (see Chapter 7). It is marketed by Titan Technologies, but is also available from Applied Analytics (see Chapter 41) as the Booster. The full Microspeed package from Applied Analytics includes the 6502C board as well as a four MHz floating point math processor, and is designed around a high speed version of the FORTH computer language.

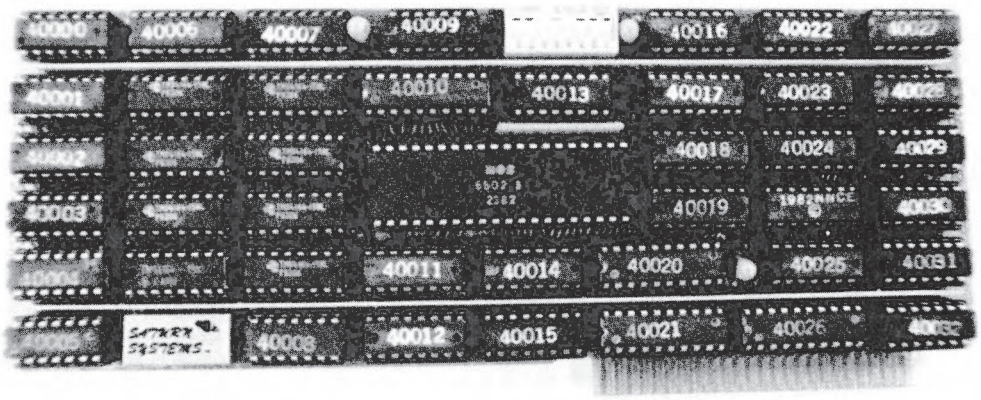


Fig. 29.7 Accelerator card from Saturn systems.

For some operations, the FORTH/6502C/Math processor combination achieves calculation speeds up to 100 times faster than standard Applesoft. There is probably no faster way to do calculations on any microcomputer, but this is certainly the fastest system for the Apple, 68000 and 8087 boards notwithstanding. Please note that this level of performance is probably meaningful only in mathematically oriented scientific tasks and complex modeling problems.

The Apple //c and the 65C02 in OXI-CMOS

As mentioned earlier, high speed in a microprocessor leads to problems with current and heat, so William Mensch of the Western Design Center undertook a complete redesign of the 6502 to take advantage of a new low current technology called Oxide Isolated Complementary Metal Oxide Semiconductor (OXI-CMOS). The result of that effort is the 65C02 which is available in one MHz, two MHz, and four MHz versions.

In the course of designing the new 65C02, Mensch gave functions to some of the unused 6502 opcodes (see Chapters 27 and 28) by adding new addressing modes and instruction types (see Table 29.1), but carefully maintained nearly all aspects of instruction timing in order to achieve complete software compatibility with most current uses of the 6502. In addition, he corrected two bugs in the original design.

Both of the bugs have to do with the way the 6502 increments its program counter as it steps across page boundaries. Normally, everything is handled correctly. However, during the execution of the JMP (indirect) instruction (see Chapter 27) and during the execution of indexed addressing (see Chapter 28), things don't work just right. In the course of reading in the indirect address for the JMP (ind) instruction, the 6502 fails to advance the high byte of the program counter as it crosses a page boundary so instead of reading the address at, for instance, two successive bytes at $45FF and $4600, it actually reads the address from the two widely separated bytes at $45FF and $4500, which generally leads to disaster. The 65C02 corrects this bug.

The other bug is much less severe but can cause accidental double reads. These are unimportant when addressing memory, but can have an effect when addressing I/O locations. The problem occurs when the 6502 calculates an indexed address which requires it to change the high order byte of the program counter as well as the low order byte. It does one read after modifying the low byte and then a second correct read after finishing with the high byte. The 65C02 does not do this.

Unfortunately, Wozniak took advantage of this quirk in the design of the Disk II controller interface, so not all versions of the new processor work properly if installed in the Apple //e. The 65C02 works perfectly well in the //c with its IWM disk controller and Disk //c drive (see Chapter 23). There have also been reports that the 65C02 will work in the //e, but not in the II/II+. It should be noted that there are 65C02s out there working just fine in Apple II+s and operating the Disk II controller, so these caveats may only apply to early production samples tested by Apple.

If you do purchase a 65C02 to install in your II/II+ or //e, and you do find that it works, you still will have some problems in programming. One reason you might want to get the 65C02 would be to write machine language software for the //c. You can use the ProDOS assembler for 65C02 codes, but the standard //e Monitor, of course, won't disassemble correctly. Programs written 65C02 code for the //c may not work on the //e since many will be designed to take advantage of the unique hardware features of the //c.

There are already two versions of the 65C02. One is manufactured by Rockwell and the other is manufactured by both NCR and GTE. You can get a Rockwell 65C02 and some more information from Southwestern Data Systems. In addition, many details about the new features are described in an article by Steven Hendrix in the December 1983 issue of Byte magazine on page 443.

The //c uses the NCR (GTE) version (see Figure 29.8). There are even more new instructions in the Rockwell version, but programs written with unique Rockwell codes will not work in the //c.

Although the 65C02 is used in the //c, its importance may be dwarfed by Mensch's more recent creations, the 65C816 and 65C802 (see Chapter 30), which can convert an Apple into a 16 bit computer with a 16 megabyte linear address range. These powerful processors have a 6502 emulation mode which functions like the 65C02. The 65C802 is fully pin compatible with the standard 6502, yet gives you the option of using a 16 bit ALU and a host of other advanced features.

Current work at the Western Design Center raises the possibility of a 32 bit processor called the 65C81632 running at 80 MHz. That device would retain compatibility with existing 6502 software through an emulation mode while also serving as the basis of a machine equivalent in power to many current mainframes.

The Motorola 6809

Motorola's own enhancement of the ancestral 6800 is the 6809, and it has been provided with a more elaborate set of registers. One key to the design differences in the Motorola 6809 is the decision to include a built-in "hardware multiply" instruction. If an eight bit number from the accumulator is multiplied by an eight bit number from memory, the result will be a 16

bit number, so a 16 bit accumulator is required. The 6809 also emphasized the execution of complex addressing instructions entirely on chip without access even to the zero page. It includes a second 16 bit index register and adds a second 16 bit stack pointer. It is very unusual to need to put more than 256 bytes in a stack, but the use of 16 bit pointers permits the stacks to be located anywhere in memory.

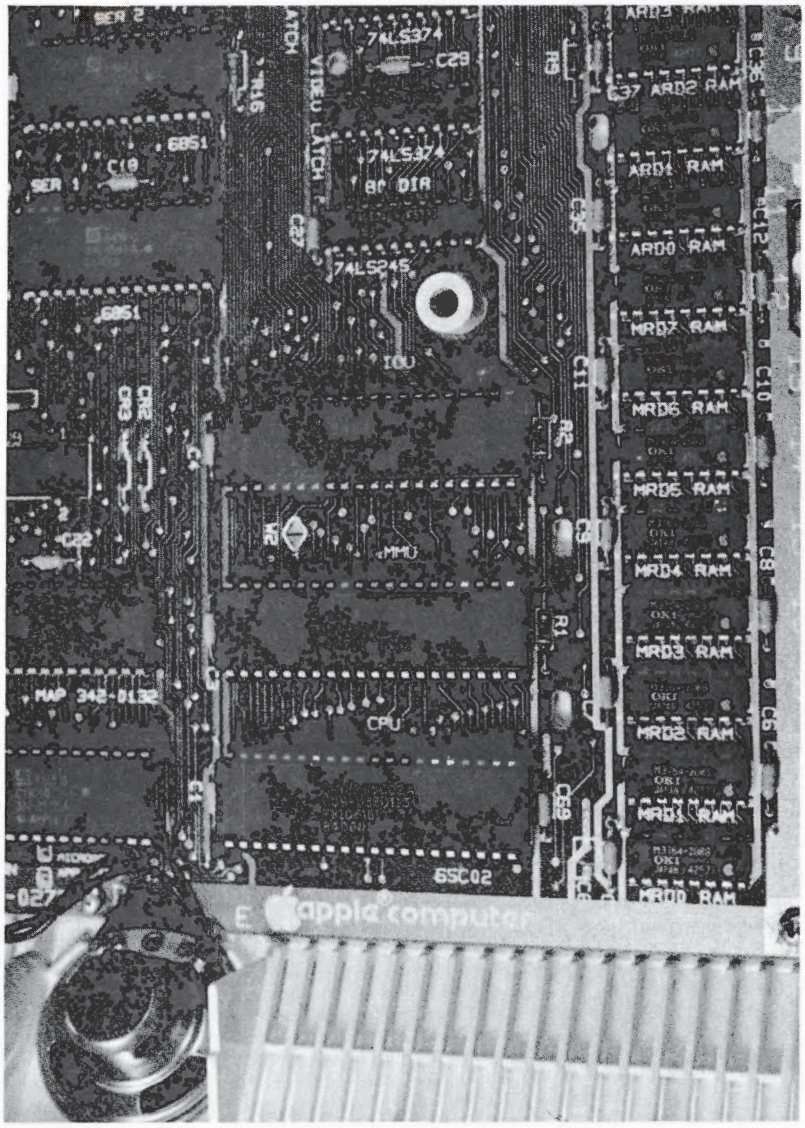


Fig. 29.8 65C02 used in the //c.

The 6809 has two stack pointers to accommodate two rather different uses of the stack which sometimes conflict. One is for scratchpad use by the programmer to hold values temporarily and to handle subroutine jumps, while the other holds register information stored during interrupts.

6809 Based Systems

Relative to other coprocessors, the 6809 doesn't have an enormous amount to contribute to the performance of the Apple. There was, however, a time before the advent of the 68000, the 8086, and the high performance upgrades of the 6502, when the Motorola 6809 seemed very attractive for high performance assembly language programming. In fact, the first design for the MacIntosh used the 6809.

The hardware multiply instruction and the versatile addressing scheme make it possible for a good, computationally oriented 6809 machine language program running at one MHz to substantially outperform equivalent programs on a one MHz 6502 or two MHz Z-80. The Mill and VitaMill from Stellation are designed to improve the running speed of Apple Pascal programs, and they also offer a fast BASIC. However, fast is a relative term. These products usually get you a 40 to 60 percent speed increase for real programs. The 6502C Accelerator (see above) can speed up the same program by 300 percent, and, if you write in UCSD Pascal for a 68000 board (see Chapter 30), you can run 800 to 1,000 percent faster.

The only other 6809 based product for the Apple is the Telidon videotex system. In videotex, there is a graphics code called NAPLPS which codes images just as ASCII codes text. Telidon had designed a videotex terminal based on the 6809 processor, so when they wanted to make a videotex system for the Apple, they followed the simple approach of taking the guts of their 6809-based terminal and simply mounting it on an Apple interface card with a 6502 interface.

Instruction Set—Alphabetical Sequence

	ADC	Add Memory to Accumulator with Carry
	AND	"AND" Memory with Accumulator
	ASL	Shift One Bit Left
	BCC	Branch on Carry Clear
	BCS	Branch on Carry Set
	BEQ	Branch on Result Zero
*	BIT	Test Memory Bits with Accumulator
	BMI	Branch on Result Minus
	BNE	Branch on Result Not Zero
	BPL	Branch on Result Plus
•	BRA	Branch Always
	BRK	Force Break
	BVC	Branch on Overflow Clear
	BVS	Branch on Overflow Set
	CLC	Clear Carry Flag
	CLD	Clear Decimal Mode
	CLI	Clear Interrupt Disable Bit
	CLV	Clear Overflow Flag
*	CMP	Compare Memory and Accumulator
	CPX	Compare Memory and Index X
	CPY	Compare Memory and Index Y
•	DEC	Decrement by One
	DEX	Decrement Index X by One
	DEY	Decrement Index Y by One
*	EOR	"Exclusive-or" Memory with Accumulator
*	INC	Increment by One
	INX	Increment Index X by One
	INY	Increment Index Y by One
*	JMP	Jump to New Location
	JSR	Jump to New Location Saving Return Address
*	LDA	Load Accumulator with Memory
	LDX	Load Index X with Memory
	LDY	Load Index Y with Memory
	LSR	Shift One Bit Right
	NOP	No Operation
*	ORA	"OR" Memory with Accumulator
	PHA	Push Accumulator on Stack
	PHP	Push Processor Status on Stack
•	PHX	Push Index X on Stack
•	PHY	Push Index Y on Stack
	PLA	Pull Accumulator from Stack
	PLP	Pull Processor Status from Stack
•	PLX	Pull Index X from Stack
•	PLY	Pull Index Y from Stack
	ROL	Rotate One Bit Left
	ROR	Rotate One Bit Right
	RTI	Return from Interrupt
	RTS	Return from Subroutine
*	SBC	Subtract Memory from Accumulator with Borrow
	SEC	Set Carry Flag
	SED	Set Decimal Mode
	SEI	Set Interrupt Disable Bit
*	STA	Store Accumulator in Memory
	STX	Store Index X in Memory
	STY	Store Index Y in Memory
•	STZ	Store Zero in Memory
	TAX	Transfer Accumulator to Index X
	TAY	Transfer Accumulator to Index Y
•	TRB	Test and Reset Memory Bits with Accumulator
•	TSB	Test and Set Memory Bits with Accumulator
	TSX	Transfer Stack Pointer to Index X
	TXA	Transfer Index X to Accumulator
	TXS	Transfer Index X to Stack Pointer
	TYA	Transfer Index Y to Accumulator

Note: • = New Instruction
* = Old Instruction with New Addressing Modes

LSD / MSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	BRK	ORA ind, X			TSB • zpg	ORA zpg	ASL zpg		PHP	ORA imm	ASL A		TSB • abs	ORA abs	ASL abs		0
1	BPL rel	ORA ind, Y	ORA * ind		TRB • zpg	ORA zpg, X	ASL zpg, X		CLC	ORA abs, Y	INC * A		TRB • abs	ORA abs, X	ASL abs, X		1
2	JSR abs	AND ind, X			BIT zpg	AND zpg	ROL zpg		PLP	AND imm	ROL A		BIT abs	AND abs	ROL abs		2
3	BMI rel	AND ind, Y	AND * ind		BIT * zpg, X	AND zpg, X	ROL zpg, X		SEC	AND abs, Y	DEC * A		BIT * abs, X	AND abs, X	ROL abs, X		3
4	RTI	EOR ind, X				EOR zpg	LSR zpg		PHA	EOR imm	LSR A		JMP abs	EOR abs	LSR abs		4
5	BVC rel	EOR ind, Y	EOR * ind			EOR zpg, X	LSR zpg, X		CLI	EOR abs, Y	PHY •			EOR abs, X	LSR abs, X		5
6	RTS	ADC ind, X			STZ • zpg	ADC zpg	ROR zpg		PLA	ADC imm	ROR A		JMP ind	ADC abs	ROR abs		6
7	BVS rel	ADC ind, Y	ADC * ind		STZ • zpg, X	ADC zpg, X	ROR zpg, X		SEI	ADC abs, Y	PLY •		JMP * ind, X	ADC abs, X	ROR abs, X		7
8	BRA • rel	STA ind, X			STY zpg	STA zpg	STX zpg		DEY	BIT * imm	TXA		STY abs	STA abs	STX abs		8
9	BCC rel	STA ind, Y	STA * ind		STY zpg, X	STA zpg, X	STX zpg, Y		TYA	STA abs, Y	TXS		STZ • abs	STA abs, X	STZ • abs, X		9
A	LDY imm	LDA ind, X	LDX imm		LDY zpg	LDA zpg	LDX zpg		TAY	LDA imm	TAX		LDY abs	LDA abs	LDX abs		A
B	BCS rel	LDA ind, Y	LDA * ind		LDY zpg, X	LDA zpg, X	LDX zpg, Y		CLV	LDA abs, Y	TSX		LDY abs, X	LDA abs, X	LDX abs, Y		B
C	CPY imm	CMP ind, X			CPY zpg	CMP zpg	DEC zpg		INY	CMP imm	DEX		CPY abs	CMP abs	DEC abs		C
D	BNE rel	CMP ind, Y	CMP * ind			CMP zpg, X	DEC zpg, X		CLD	CMP abs, Y	PHX •			CMP abs, X	DEC abs, X		D
E	CPX imm	SBC ind, X			CPX zpg	SBC zpg	INC zpg		INX	SBC imm	NOP		CPX abs	SBC abs	INC abs		E
F	BEQ rel	SBC ind, Y	SBC * ind			SBC zpg, X	INC zpg, X		SED	SBC abs, Y	PLX •			SBC abs, X	INC abs, X		F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

NOTE: • = New Instruction
* = Old Instruction with New Addressing Mode

Microprocessor Op Code Table

Table 29.1a Comparison table for new 65C02 instruction types and available addressing modes.

Operational Codes, Execution Time, and Memory Requirements

MNE-MONIC	OPERATION	IMMEDIATE OP	n	#	ABSOLUTE OP	n	#	ZERO PAGE OP	n	#	(4) IMPLIED OP	n	#	(IND, X) OP	n	#	(1) (IND), Y OP	n	#	ZPG,X OP	n	#	(1) ABS,X OP	n	#	(1) ABS,Y OP	n	#	RELATIVE (2) OP	n	#	INDIRECT OP	n	#	ZPG,Y OP	n	#	PROCESSOR STATUS CODE 7 6 4 3 2 1 0 N V B D I Z C	MNE-MONIC
ADC	A + M + C → A (3)	69	2	2	6D	4	3	65	3	2				61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3				72	5	2				N V • • • Z C	ADC
AND	A∧M → A	29	2	2	2D	4	3	25	3	2				21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3				32	5	2				N • • • • Z •	AND
ASL	C ← [7 0] ← 0				0E	6	3	06	5	2	0A	2	1							16	6	2	1E	6	3													N • • • • Z C	ASL
BCC	BRANCH IF C=0 (2)																												90	2	2							• • • • • • •	BCC
BCS	BRANCH IF C=1 (2)																												B0	2	2							• • • • • • •	BCS
BEQ	BRANCH IF Z=1 (2)																												F0	2	2							• • • • • • •	BEQ
BIT	A ∧ M (5)	89	2	2	2C	4	3	24	3	2										34	4	2	3C	4	3													M₇ M₆ • • Z •	BIT
BMI	BRANCH IF N=1 (2)																												30	2	2							• • • • • • •	BMI
BNE	BRANCH IF Z=0 (2)																												D0	2	2							• • • • • • •	BNE
BPL	BRANCH IF N=0 (2)																												10	2	2							• • • • • • •	BPL
BRA	BRANCH ALWAYS(2)																												80	2	2							• • • • • • •	BRA
BRK	BREAK										00	7	1																									• • 1 0 1 • •	BRK
BVC	BRANCH IF V=0 (2)																												50	2	2							• • • • • • •	BVC
BVS	BRANCH IF V=1 (2)																												70	2	2							• • • • • • •	BVS
CLC	0 → C										18	2	1																									• • • • • • 0	CLC
CLD	0 → D										D8	2	1																									• • • 0 • • •	CLD
CLI	0 → I										58	2	1																									• • • • 0 • •	CLI
CLV	0 → V										B8	2	1																									• 0 • • • • •	CLV
CMP	A-M	C9	2	2	CD	4	3	C5	3	2				C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3				D2	5	2				N • • • • Z C	CMP
CPX	X-M	E0	2	2	EC	4	3	E4	3	2																												N • • • • Z C	CPX
CPY	Y-M	C0	2	2	CC	4	3	C4	3	2																												N • • • • Z C	CPY
DEC	DECREMENT				CE	6	3	C6	5	2	3A	2	1							D6	6	2	DE	6	3													N • • • • Z •	DEC
DEX	X-1 → X										CA	2	1																									N • • • • Z •	DEX
DEY	Y-1 → Y										88	2	1																									N • • • • Z •	DEY
EOR	A ⊻ M → A	49	2	2	4D	4	3	45	3	2				41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3				52	5	2				N • • • • Z •	EOR
INC	INCREMENT				EE	6	3	E6	5	2	1A	2	1							F6	6	2	FE	6	3													N • • • • Z •	INC
INX	X + 1 → X										E8	2	1																									N • • • • Z •	INX
INY	Y + 1 → Y										C8	2	1																									N • • • • Z •	INY
JMP	JUMP TO NEW LOC				4C	3	3							7C	6	3																6C	6	3				• • • • • • •	JMP
JSR	JUMP SUB				20	6	3																															• • • • • • •	JSR
LDA	M → A	A9	2	2	AD	4	3	A5	3	2				A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3				B2	5	2				N • • • • Z •	LDA
LDX	M → X	A2	2	2	AE	4	3	A6	3	2																BE	4	3							B6	4	2	N • • • • Z •	LDX
LDY	M → Y	A0	2	2	AC	4	3	A4	3	2										B4	4	2	BC	4	3													N • • • • Z •	LDY
LSR	0 → [7 0] → C				4E	6	3	46	5	2	4A	2	1							56	6	2	5E	6	3													0 • • • • Z C	LSR
NOP	NO OPERATION										EA	2	1																									• • • • • • •	NOP
ORA	A∨M → A	09	2	2	0D	4	3	05	3	2				01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3				12	5	2				N • • • • Z •	ORA
PHA	A→Ms S-1→S										48	3	1																									• • • • • • •	PHA
PHP	P→Ms S-1→S										08	3	1																									• • • • • • •	PHP
PHX	X→Ms S-1→S										DA	3	1																									• • • • • • •	PHX
PHY	Y→Ms S-1→S										5A	3	1																									• • • • • • •	PHY
PLA	S+1→S Ms→A										68	4	1																									N • • • • Z •	PLA
PLP	S+1→S Ms→P										28	4	1																									N V 1 D I Z C	PLP
PLX	S+1→S Ms→X										FA	4	1																									N • • • • Z •	PLX
PLY	S+1→S Ms→Y										7A	4	1																									N • • • • Z •	PLY
ROL	[C ← 7 0 ←] C				2E	6	3	26	5	2	2A	2	1							36	6	2	3E	6	3													N • • • • Z C	ROL
ROR	[→C→ 7 0]				6E	6	3	66	5	2	6A	2	1							76	6	2	7E	6	3													N • • • • Z C	ROR
RTI	RTRN INT										40	6	1																									N V 1 D I Z C	RTI
RTS	RTRN SUB										60	6	1																									• • • • • • •	RTS
SBC	A-M-C̄→A (3)	E9	2	2	ED	4	3	E5	3	2				E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3				F2	5	2				N V • • • Z C	SBC
SEC	1 → C										38	2	1																									• • • • • • 1	SEC
SED	1 → D										F8	2	1																									• • • 1 • • •	SED
SEI	1 → I										78	2	1																									• • • • 1 • •	SEI
STA	A → M				8D	4	3	85	3	2				81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3				92	5	2				• • • • • • •	STA
STX	X → M				8E	4	3	86	3	2																											96	4	2	• • • • • • •	STX
STY	Y → M				8C	4	3	84	3	2										94	4	2																	• • • • • • •	STY
STZ	00 → M				9C	4	3	64	3	2										74	4	2	9E	5	3													• • • • • • •	STZ
TAX	A → X										AA	2	1																									N • • • • Z •	TAX
TAY	A → Y										A8	2	1																									N • • • • Z •	TAY
TRB	A̅∧M → M (6)				1C	6	3	14	5	2																													• • • • • Z •	TRB
TSB	A∨M → M (6)				0C	6	3	04	5	2																													• • • • • Z •	TSB
TSX	S → X										BA	2	1																									N • • • • Z •	TSX
TXA	X → A										8A	2	1																									N • • • • Z •	TXA
TXS	X → S										9A	2	1																									• • • • • • •	TXS
TYA	Y → A										98	2	1																									N • • • • Z •	TYA

Notes

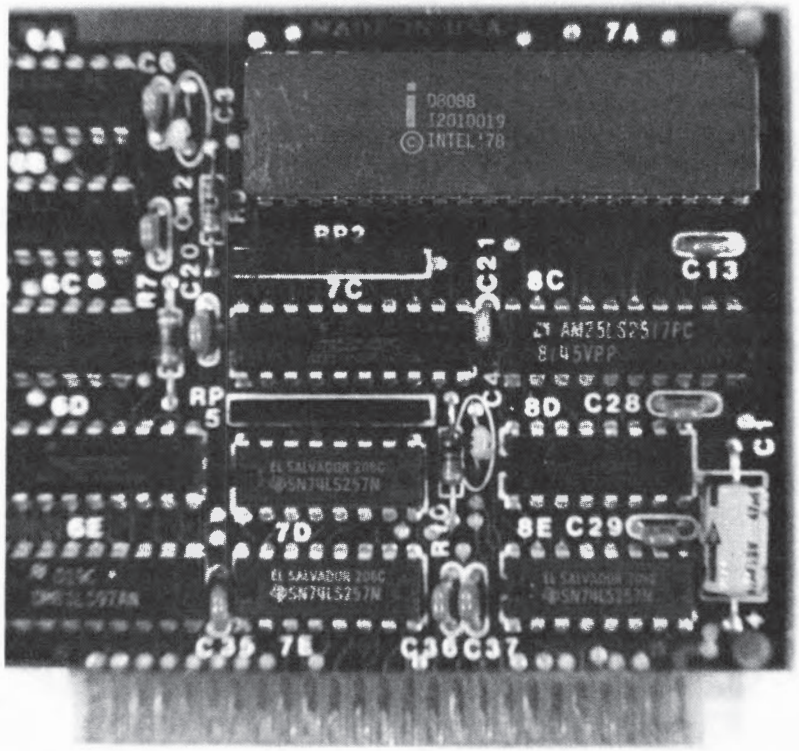
1 Add 1 to "n" if page boundary is crossed, except STA and STZ
2 Add 1 to "n" if branch occurs to same page. Add 2 to "n" if branch occurs to different page.
3 Add 1 to "n" if decimal mode.
4 Accumulator address is included in Implied address
5 "N" and "V" flags are unchanged in immediate mode
6 "Z" flag indicates A∧M result (same as BIT instruction)

X Index X
Y Index Y
A Accumulator
M Memory per effective address
Ms Memory per stack pointer

+ Add
- Subtract
∧ And
∨ Or
⊻ Exclusive or

n No. Cycles
No. Bytes
M₆ Memory Bit #6
M₇ Memory Bit #7

Table 29.1b Opcodes, execution times, and instruction lengths for 65C02.


8A
7A
C6
C3
C20
C21
8C
7C
C13
RP5
8D
C28
6E
7D
8E
C29
7E
C36
C37

Chapter 30

Microprocessors: The Fourth Generation and Beyond

The Fourth Generation

In 1979, Intel, Zilog and Motorola all introduced microprocessors which had 16 bit Arithmetic and Logic Units (ALUs), and which could take advantage of the arrival of the new 64K dynamic RAM chips. The expectation that 64K bytes of RAM would soon cost less than $100 called for larger address ranges, and it also made it possible to consider reorganizing external memory into 16 bit words instead of eight bit bytes. When memory is organized strictly as 16 bit "words," it takes 128K bytes of RAM to store 64K of characters, so memory must be cheap to support this.

The Intel 8086 extends the tradition of the 8080/8085 line in that a substantial part of the design effort was devoted to limiting time spent accessing external memory. The Motorola 68000, on the other hand, relies on very high speed operation and high speed memory devices for improved performance. The general concensus in 1979 was that the 8086 seemed better suited to microcomputers, and that the 68000 was a bit too brawny. The advent of high speed 256K RAM chips in 1983, however, has shifted attention towards the 68000 for high performance microcomputer designs.

The selection of the somewhat underpowered 8088 by IBM for their personal computer, however, has resulted in the development of a huge software base for the 8088 design. The IBM PC was originally released with just 16K of RAM memory (really), so their choice of a low powered 16 bit processor seems to reflect an initial misjudgement of the potential for powerful microcomputers. The 8088 is a version of the 8086 with an eight bit external data path which essentially forbids the use of 16 bit data words as an option for organizing memory for calculations.

This selection has led to the paradoxical situation of programs running faster on a Z-80 machine designed in 1976 than they do on the IBM 8088 machine designed in 1981. The newest versions of the IBM PC (the PC 3270 and the XT/370) include what are effectively three add-on 68000s, quite a far cry from a lone 8088. Meanwhile, Apple, which has released two 68000 based machines, is looking to the 32 bit processors of 1983 and 1984, such as the WDC 6581632.

The 8086 and 8088

The choice of attempting to minimize memory access has led to some humbling problems for 8086 based designs. A three byte instruction can contain a one byte opcode and a 16 bit address. One byte opcodes can describe only 256 different instructions, so the 8086/8088 instruction set retains all the peculiarities, quirks and limitations of eight bit microprocessors. A 16 bit address can only refer to 64K bytes of memory, but Intel chose not to include a third address byte with every instruction. To increase the memory range of the microprocessor, Intel implemented a segmentation scheme. The great bulk of language interpreters and commercial software written for the 8088/8086 has tended to work mostly within a single 64K "segment."

Each address which the 8086 announces is a 20 bit address, so there is an address range of over one megabyte. However, the address space of the 8086 should be viewed as containing four active segments each containing 64K locations. There are four 16 bit "segment registers" (see Figure 30.1) in the 8086 which determine the beginning locations of each of the 64K segments. The four segment registers are called the Data, Code, Stack, and Extra segment registers. The four segments can be positioned as windows into any 64K range within the one megabyte address space.

There are five additional 16 bit registers which are used for address calculation. Once a segment register is set, a particular location in memory is addressed by selecting a value in one of the address registers and adding it to the value in one of the segment registers. The contents of the segment register are first shifted left four places to generate an effective 20 bit address with four trailing zeros and then added to the address register value. For instance, in hexadecimal this would look as follows: Stack Segment equals $8333, Stack Pointer equals $2222, so the Stack Segment is shifted to yield $83330, and then added to the value of $2222, producing a result of $85552 as the actual 20 bit address.

Careless changes in segment register values can cause havoc, so most programmers set up one segment for the stack, one for the program, one for user data, and one for the operating system and just leave them. Since many IBM PC owners have just 64K of RAM, the segments are often pointed as slightly different locations within a single 64K space. However, as 256K of RAM became more standard on IBM PCs, the segment system came to be used to load several large program and data sections into different areas of memory and then to switch rapidly among them by manipulating the segment registers.

The other innovation in the 8086 (which did not appear elsewhere until the 1984 release of the Motorola 68020) is the inclusion of a six byte "instruction que" in the 8086 (and a four byte que in the 8088). The 8086 microprocessor is divided into two parts, one called the Execution Unit (EU) and one called the Bus Interface Unit (BIU), and these two operate independently and asynchronously. The execution unit goes about doing math and calculating addresses, while the bus interface unit holds the segment addresses, takes care of linear incrementing of the program counter (called the instruction pointer) and attempts to fetch instructions independent of their execution.

The BIU assumes that program instructions will be in a linear sequence without branches and it uses the instruction pointer and the code segment register to fill up its six byte instruction que register section with the next six instructions. Whenever the EU finishes with an instruction, it pulls the next one off the que and tells the BIU to get some more. This theoretically prevents the EU from ever having to wait for an instruction fetch. This is a nice design concept, but it is difficult to show that it adds much to processing speed.

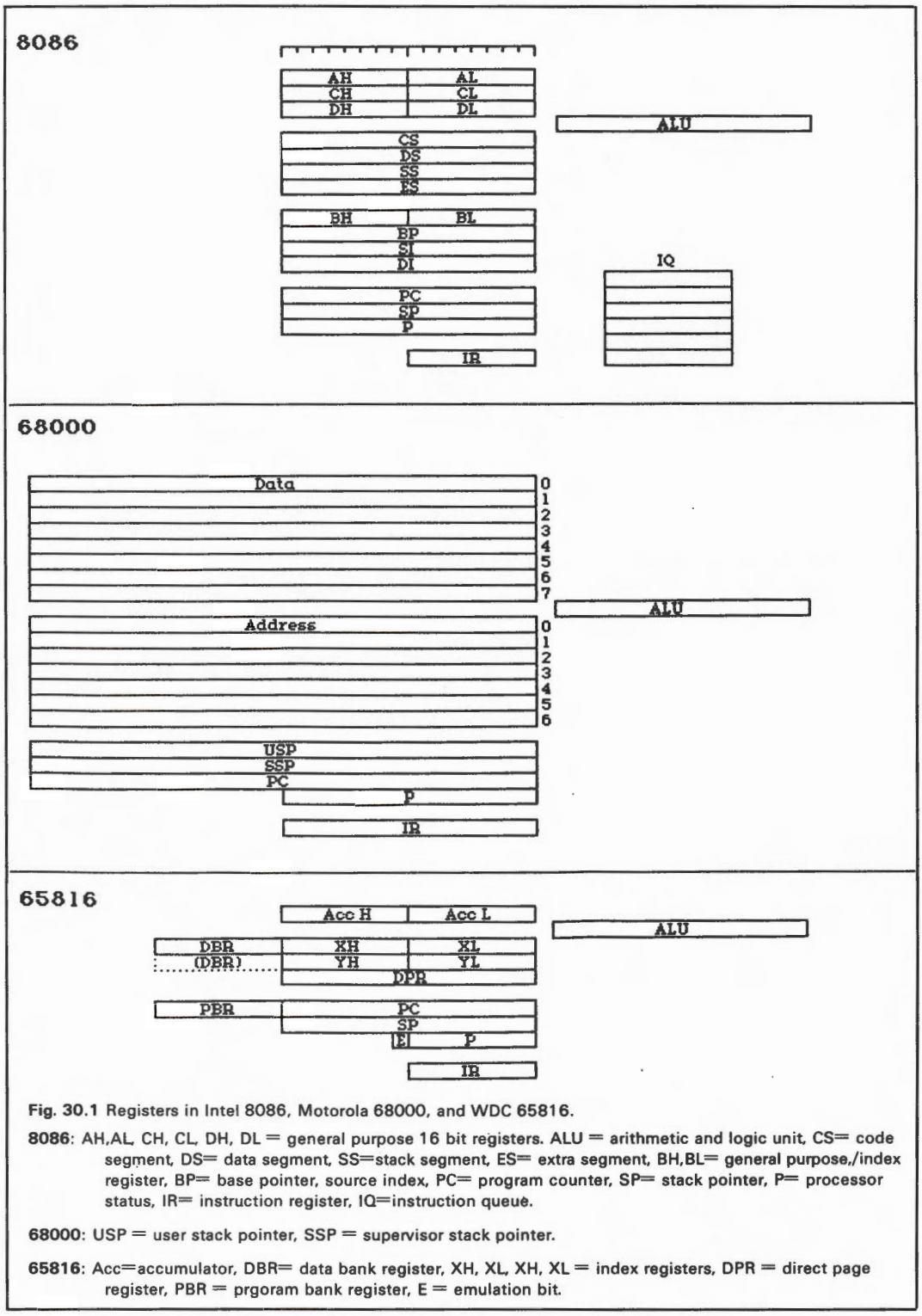


Fig. 30.1 Registers in Intel 8086, Motorola 68000, and WDC 65816.

8086: AH,AL, CH, CL, DH, DL = general purpose 16 bit registers. ALU = arithmetic and logic unit, CS= code segment, DS= data segment, SS=stack segment, ES= extra segment, BH,BL= general purpose,/index register, BP= base pointer, source index, PC= program counter, SP= stack pointer, P= processor status, IR= instruction register, IQ=instruction queue.

68000: USP = user stack pointer, SSP = supervisor stack pointer.

65816: Acc=accumulator, DBR= data bank register, XH, XL, XH, XL = index registers, DPR = direct page register, PBR = prgoram bank register, E = emulation bit.

The 8086 includes a 16 bit multiply command which can execute in about 40 microseconds in a five megahertz 8088, such as the one installed in IBM PC. This involves using the same sort of simple shift and add ALU system as traditional microprocessors, but excludes much of the memory access. The operation takes about 150 clock cycles. This is about a tenfold improvement over the same calculation with a four megahertz Z-80A. However, the 8088 does not handle 16 bit operands and 32 bit results very well. Intel actually advises programmers to use shift and add instructions in most situations.

The 8087 Floating Point Math Processor

The 8086 or 8088 can be enhanced by the addition of an 8087 floating point math processor. The 8087 costs about six times as much as an 8088, and so can add many hundreds of dollars to the cost of a system. Therefore, no major microcomputer manufacturer has been shipping 8088 systems with 8087s installed, and so the potential base for commercial software is small. There are versions of FORTRAN and C which can use the 8087 when installed in some systems, so it can provide a great improvement in mathematical power. The eight bit data bus of 8088 systems tends to choke off much of the advantage, however. S-100 microcomputers such as the Lomas Lightning One, which include an 8086 and an 8087 actually can achieve very impressive numeric results.

The 8087 captures all instructions in parallel with the 8086, but when an escape sequence in the instruction series is detected, the 8087 takes over. The 8087 uses numbers which have a 14 bit exponent and a 64 bit "significand." This works out to a precision of 18 decimal digits. For most operations, such as division and exponentiation, there is a speed increase of about 100 times for the actual calculation. Further, it is fairly difficult to carry out floating point math with very large numbers by brute force programming on an 8088.

The 80186 and the 80286

Since 1979, Intel has been working on a variety of updates of the original 8086/8088 (see the Intel Microprocessors and Peripherals Handbook for a complete overview) and two of these have appeared in microcomputers in 1983. The 80186/80188 is directed towards low end systems. It preserves all of the features of the 8086, but it relies on advances in fabrication technology to pack additional circuitry onto the chip, much in the same way that the 8085 improved upon the 8080.

The 80188 provides no increase in absolute performance, but it replaces 10 or 15 chips with the single 80188 package and this reduces power requirements, space requirements and cost. The major additional features are DMA (see Chapter 27) and interrupt handling systems, as well as programmable chip select and counter/timer functions.

The 80286, however, is a major upgrade of the 8086. The most important advance is the inclusion of a memory management system on board. The 80286 has a real address mode in which it behaves just like a standard 8086. However, it also has a virtual memory mode in which the process of assembling effective addresses is changed. In virtual memory mode, the segment registers are used as 16 bit pointers to segment descriptors in external memory, and these are used to construct 24 bit addresses instead of the 20 bit addresses used by the 8086. The result is a physical address space of 16 megabytes instead of one megabyte.

Further, a new control section on the 80286 is able to make prioritized decisions about when it can replace a block of memory with new program or data information from external storage. This system provides for the management of one gigabyte, the 16 megabyte address space is treated as a window into a space which is 64 times as large. The 80286 is important for the continued viability of the 8086 family because, for instance, the software used on Apple's Lisa system is over two megabytes in size. This completely overwhelms the capabilities of an 8086 or 8088.

Memory management systems are available for several other advanced microprocessors, and the 80286 generally receives fairly high marks. However, it remains to be seen whether the fundamental 8086 processor at the heart of the system will provide a sufficiently powerful processing environment to make adequate use of that much memory.

Most system designers interested in using powerful memory management systems are looking for newer processors which can provide 32 bit ALUs and which are not bound by the 8086's comparatively tiny instruction set. The 32 bit Intel 80386 may alleviate some of these problems, but the Motorola 68000 and the National Semiconductor 16032 and 32032 chips have come to market far ahead of the 80386, and the long term future of the segmented memory approach of the 8086 is not yet clear.

Apple 8088s, 8086s and PC Compatibility

The 8088 doesn't have too much to add to the computational power of the Apple, and doesn't provide the range of software available by adding a Z-80. What you can hope to get with an 8088 based add-on is access to a small number of very powerful and memory intensive business packages developed for the IBM PC. These are all packages which are designed for 128K and 256K of RAM, usually for MS-DOS 2.0 (see Chapter 35). Further, the ability to add IBM PC compatibility to a well-equipped Apple greatly extends the value of the original purchase. This is particularly true in business where the great majority of new packages are being written first for the IBM PC rather than for the Z-80.

As a result of these considerations, the first question you should ask when you select an 8088 or 8086 based peripheral is: To what extent is this device compatible with the IBM PC? If you've ever shopped for an IBM PC compatible machine, you know this is a tricky question. For complete compatibility, a machine must have the PC's 10 function keys, all of the PC's graphics modes, PC disk format, and IBM ROMs. Because complete compatibility is not really legal, most PC software is released in at least two versions, one for the PC and one in generic MS-DOS form. However, most of the software available in generic MS-DOS form is also available for CP/M (see Chapter 29), so this is no reason to buy an 8088 based product for the Apple.

The next slant in the compatibility question has to do with machines such as the TI Professional or Rana 8086/2, which are quite close to the PC in operation, but which offer substantial additional features. To take advantage of such features, software vendors such as Microsoft and Lotus must write special versions of their programs for the machine.

With these considerations in mind, you can consider the features of four 8088 or 8086 based products for the Apple. The Rana 8086/2 is a complete PC compatible system with support for its special features from several major software firms. The other three products are single boards which don't provide any level of PC compatibility. They can be used for generic MS-DOS software, but you need to see to special provisions to deal with disk incompatibility. Of these, the Coprocessor 8088 from PCPI is the poorest choice because it comes with just 64K

of RAM, has no provision for expansion, and is so poorly documented that even assembly language programmers will find it less than useful.

The MetaCard from Metamorphic Systems and the AD8088 from ALF are both board level 8088 products which have some merits as a low cost means of getting some MS-DOS compatibility. The MetaCard has space for 128K of RAM on the card and provides for expansion to a second board with an additional 256K and an 8087. There is no provision for disk compatibility, so you have to get generic MS-DOS or CP/M-86 software on specially prepared disks either through Metamorphic Systems or from some of the larger software supply houses.

ALF's AD8088 card is the best choice for a board level product because this company has taken several steps to help make sure it is a useful product. First, even if you never get any MS-DOS software, the AD8088 is designed to act as true coprocessor to support Applesoft BASIC (see Chapter 41) and Apple Pascal 1.1. In both languages, it can intervene in the execution of a program and speed up mathematically intensive work. It's not that the 8088 is that much faster than the 6502 in general, but by selecting those chores at which the 8088 has the greatest advantage, the ALF systems coordinates the two processors to achieve speed increases of up to three or four times for some programs.

If you are interested in some degree of PC compatibility, ALF will sell you their DC3 disk controller which will let you read and write IBM PC disks. The DC3 goes into an Apple slot, but you have to buy and interface your own drives. The AD8088 comes with just 64K of RAM on board but, like the Coprocessor 8088 and the MetaCard, it can use the Apple's own RAM to boost the total to 128K. The ALF card also has an expansion system for connection to a second card which can hold an 8087, 128K of RAM, and/or a special graphics subsystem which helps the 8088 make better use of the Apple's own video graphics system.

The Rana 8086/2

Not only is the Rana 8086/2 a completely useful MS-DOS 2.0 system, it but may be the best PC compatible on the market as well as being one of the best enhancements you can add to an Apple II (see Figure 30.2). To begin with, it includes two slim line 320K drives (360K under MS-DOS 2.0) which you can use with Apple ProDOS, CP/M and Pascal. Therefore, you get two high density floppies before you even begin to consider PC compatibility.

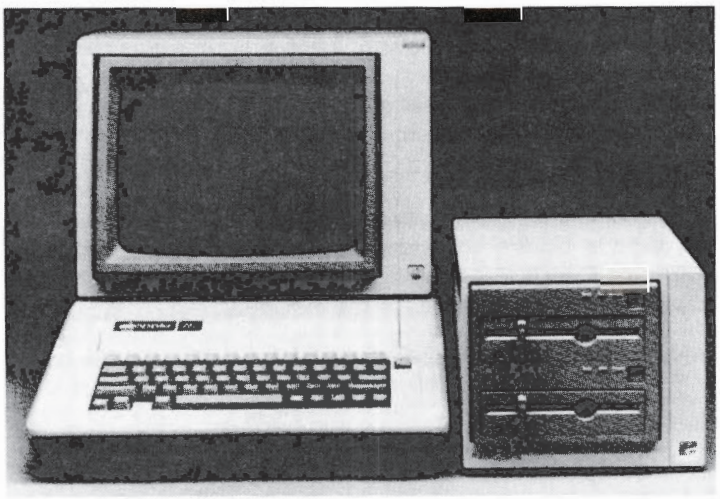


Fig. 30.2 Rana 8086/2. The box contains an 8086 with 256K of RAM, 640 X 400 graphics, and two 360K drives.

The system comes with 256K of RAM installed, expandable to 512K, an 8086 processor configured as a full 16 bit machine, and a socket for the 8087. It has a full implementation of all of the IBM PC video graphics modes including dual port video RAM which IBM uses to simplify the timing of video refresh. In addition, it offers a high resolution 320 by 400 dot color graphics output and ultra high res 640 by 400 dot monochrome graphics capability. There are two pages of video memory so you can do much more elaborate graphics than on a standard PC.

PC Compatible Disk Drives

The disk drive system is a marvelously clever bit of electronics which reflects the experience of Rana as a leading manufacturer of high density Apple compatible drives (see Chapter 24) and of Rana Engineering Vice President Don Burtis (who also designed the enormously popular Microsoft Z-80 SoftCard). The clocked encoding system for magnetic bits on an Apple disk is very different from the encoding system used in the IBM PC. Rana has therefore built two completely different sets of drive control circuitry to operate the drives.

The controller can detect which kind of disk has been inserted and activate the appropriate electronic subsystem. If you put in an Apple encoded disk, then the whole system behaves as if there is nothing there but a high density Rana disk drive with a standard Apple controller. This is crucial because it means you can buy an Apple //e without drives or controller and use the Rana subsystem to boot all your Apple DOS, ProDOS, CP/M and Apple Pascal 1.1 disks.

However, if an IBM PC style disk is inserted, the data is fed directly into the 8086/2 which then takes control of the entire system. From the 6502's point of view, when this happens, the interface card in the Apple bus transforms itself from a disk controller into a parallel port communications connection to an 8086 computer. This is important because now the 8086/2 can send requests to the 6502, asking it to send characters to your printer, modem, voice synthesizer or whatever.

Software Support

More important than the technical details, Microsoft has prepared a version of Windows (see Chapter 37) to take advantage of the special features of the 8086/2, and Lotus has a version of 1-2-3 prepared for the device. In the PC compatible market, there is great hesitation to attempt technical innovations not included by IBM in its PC. The use of an 8086 and extended video modes in this machine reflects the belief on Rana's part that the Apple-8086/2 system can be a powerful computing system in its own right.

The Motorola 68000, 68010 and 68020

Among the most important design choices for the 68000 were the decisions to use two byte opcodes and to provide a linear, contiguous 16 megabyte address space. The 8086 is limited to an instruction set no larger than 256 instructions as in the 6502 or Z-80, but the 68000 can use a much more elaborate instruction set with many hundreds of instructions, up to 64,000 in fact. The additional instructions are based around a few simple types, but there are virtually no limitations on which combination of instruction type, data type, and addressing mode are used to construct a given instruction. This relieves the programmer of much of the tedium of remembering special cases for various instructions and leads to more open, expressive coding.

The 68000 actually assembles 32 bit addresses which can access up to four gigabytes of physical addresses, although only 24 bits of each address are actually available on address pins. The

68010 uses the 24 bit address in a memory management scheme to "fold" the full 16 megabyte address range out onto disk drives, so that only part of the "logical" address space is actually in the, i.e., one megabyte of RAM at any given time.

The more advanced 68020 makes use of the full 32 bit address in a four gigabyte virtual memory scheme. This is a high performance memory management scheme relative to the 80286 since all the addresses are calculated directly and completely on board the 68020 without memory accesses or segment calculation. The 68000 itself can address a linear contiguous space of 16 megabytes without resort to the complex segmentation schemes used by 8086 type processors. In addition, the 68020 has an instruction queue much as in the BIU of the 8086 (see above) and an expanded instruction set.

Computation with the 68000 and 68881

In addition, although the 68000 uses a 16 bit ALU like the 8086, it has eight 32 bit registers for data and seven 32 bit registers for address construction. This greatly reduces the complexity of mathematical tasks since there is plenty of room on the chip for intermediate results as well as for the 32 bit results of 16 bit multiplication. While the five MHz 8088 requires 40 microseconds for a 16 bit multiplication, a 12.5 MHz 68000 can do a 32 bit multiplication with a 64 bit result in just 60 microseconds.

The mathematical power of the 68000 far exceeds the 8086, and reduces the need for a floating point coprocessor such as the 8087 in all but the most numerically oriented applications. However, together with the 68881 floating point processor, the computational power of a 68000 based system dwarfs the 8088/8087 combination. Intel based systems that use the 8086 can do a bit better since they can use 16 bit data to feed the processor. Speed is also very important in "number crunching" tasks, and, in 1983, the 8086/8087 chip set could not be operated faster than five megahertz, while the 68000/68881 was available at 15 megahertz. The 68020 has enhanced onboard facilities for executing mathematical operations, but it is also designed with a built-in interface for the 68881.

High Performance 68000 Boards for the Apple II

The 68000 is available on circuit boards for the Apple in eight MHz, 10 MHz, 12.5 MHz and 14 MHz versions, but only expensive "100 nanosecond" dynamic RAM chips can operate fast enough to keep up with the 12.5 and 14 MHz chips. Static RAMs which are a little more expensive and which take up more space on a board can keep up. The 68020 can run at 22 megahertz, but this is only useful in systems with very expensive support chips.

The 68000 is provided with a signal line called DTACK, which can be turned on by slower memories to tell the 68000 to wait while they catch up. Digital Acoustics makes an Apple II interface for two versions of the high speed 68000. One of these can support a megabyte of the less expensive dynamic RAMs, and it is called the DTACK Grande. Digital Acoustics also sells one with static RAMs that roars along full speed at 12.5 MHz. In this version, the DTACK line never has to get turned on so it is simply connected permanently to ground. This board is called the DTACK Grounded and it can truly expand your Apple's mind.

Another problem at these high operating speeds is stray capacitance (see Chapter 13) on the circuit board itself. This has the physical effect of slowing signal transitions (going from 1 to 0, etc.) but practically, this means that some boards work and some don't, and others only

work most of the time. The printed circuit board technology on which microcomputers have always relied starts to get overwhelmed at these speeds.

The Saybrook 68000 board (see Figure 30.3) uses an advanced circuit board technology called "multiwire" in which individual insulated wires are used to make connections, thus greatly reducing capacitive interactions. The wires are cast in epoxy so that the whole thing looks just like a standard circuit board. Multiwire has been used in high speed, very high performance military electronics, but the Saybrook 68000 may be the first application of this technology in consumer electronics.

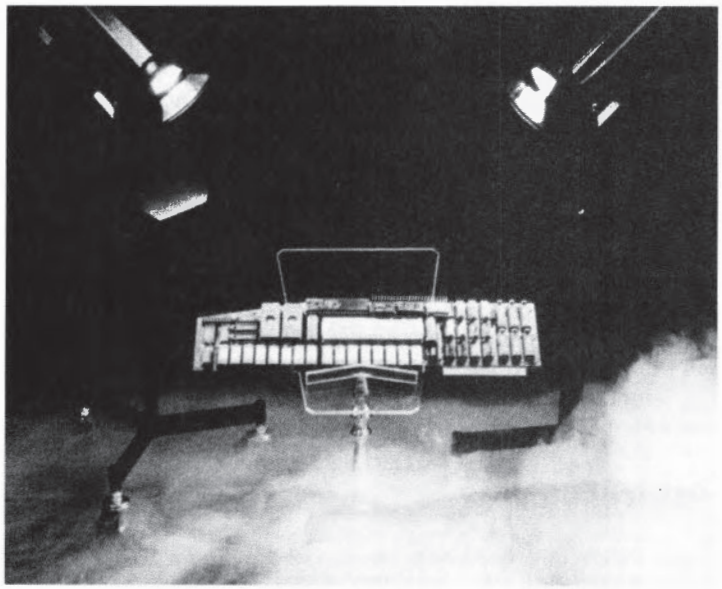


Fig. 30.3 Saybrook 68000 board from Analytical Engines.

Four other companies are now manufacturing 68000 based boards for the Apple II, (ETC, IBS, Stellation, and Acorn) all of which are practical for the programmer who will be working in UCSD Pascal, Applesoft BASIC or FORTRAN. However, folks with dedicated applications in signal processing or graphics, and with substantial enthusiasm for machine language programming and a little bit of hardware fiddling, should know that the unadorned DTACK Grounded is the speed/cost champion.

The Saybrook board is offered in an eight megahertz version, but also in a high performance 12.5 megahertz full speed version (no "wait states"). The Saybrook may be expanded to include a direct hard disk connection, and is the board to watch for a 68881 by 1985. Saybrook can deliver top of the line performance along with a rich selection of operating systems (UCSD Pascal IV.1, CP/M 68K, UNIX Version 7, FORTH), several high level languages (FORTRAN, Pascal, BASIC, Applesoft BASIC), and even applications such as word processing, spreadsheets and database software.

The Saybrook board has 128K bytes of RAM on board made up from 64K bit RAM chips, but these can be replaced with the new 256K bit RAM chips thus bringing the on board total to 512K bytes. It is supplied with an external power supply and has provisions for attaching an expansion bus directly to the card. The ETC system involves two boards and comes with a total of 256K of RAM installed, all using 64K bit chips.

There is not much available now in the way of commercial software to run on the 68000, although Digital Research has released CP/M 68K for it. Further, although Apple is using the 68000 in the Lisa and the MacIntosh, and IBM is using it in its advanced PCs and its Instrument Computer, the 68000's software history is likely to be more like that of the 6502 than the Z-80. This means that most 68000 software for Apple and IBM machines is likely to be dependent on external system hardware, and therefore not be portable to other computer systems using the same chip. However, the manufacturers of the Saybrook board have been able to modify some of the small number of applications written in UCSD Pascal IV.1 to run on their 68000.

The 68000 is very popular for programming in UCSD Pascal IV.1 because although this is an extremely slow language on most computer systems, programs in this language will run quite crisply on a 68000. The overall performance of commercial software using a slow language on a fast processor won't be much better than what is available by using a fast language on a slower processor, such as the 6502.

The WDC 65816 and 65802

The architecture of the Western Design Center's 65816 is an interesting departure from the trends followed by Motorola, Intel and other microprocessor design firms. Arriving as it does in 1984, it reflects the impending fusion of microcomputer and mainframe architecture. It is the first microprocessor other than the DEC Micro Vax1 which has a "cache" oriented memory design, yet it also differs from all other advanced microprocessors in providing a software selectable emulation mode of a well supported 8 bit microprocessor, the 6502.

Options for 24 Bit Addresses

Like the fourth generation processors (8086, 68000, Z8000), the 65812 provides a 16 bit design based on the spirit of an older processor, in this case the 6502. Along these lines, all of the 6502's 8 bit registers are widened to 16 bits, including the ALU (see Figure 30.1). In addition, the 65816 provides two means of generating 24 bit addresses for a 16 megabyte address range.

In the first option, you can use a data bank register and a program bank register to set up a segmented memory scheme much as in the 8086. The bank register provides the first eight bits of the address and the instruction identifies the remaining 16 bits. The data bank register is more sophisticated than the 8086 segment registers in that it will increment when an address in the X or Y index register overflows. This prevents the kind of rollover that occurs in 8088/8086 programming. The other mode provides for a linear, unsegmented 16 megabyte range by letting you expand your instructions to include three address bytes.

New Registers and Control Lines

In addition to expanding upon the original 6502 registers, the design is also enhanced by a new 16 bit direct page register which effectively lets you position the zero page anywhere in the first 64K of RAM. Thus you can use the compact zero page instructions (now called direct page instructions) anywhere in the first 64K by adjusting the direct page register before issuing the instruction. Similarly, since the stack pointer is an unbanked 16 bit register, the stack can be placed anywhere in the first 64K. The ultimate effect of these changes suggests the design of a machine with a very large amount of memory but in which the entire first 64K ($00 Bank) plays the special role once reserved for the zero page in 6502 based designs.

The advanced features of the design are based on five new control lines: VPA (Valid Program Address), VDA (Valid Data Address), ABORT, VP (Vector Pull) and ML (Memory Lock). The memory lock signal is used for multiprocessor systems to inform other processors that the 65816 is performing a "read, modify, write" instruction such as a decrement (DEC) of a data value in memory and thus cannot relinquish control of the bus. Vector Pull is the 65816's way of announcing that it is starting to handle an interrupt, which is important information for designing complex interrupt handling systems.

Memory Management and High Speed Cache Architecture

More important are the VPA, VDA and ABORT lines. These can be used to implement a memory management system and/or a cache memory architecture. ABORT is somewhat like NMI or IRQ on the 6502, except that it halts the execution of an instruction immediately without allowing the 65816 to progress through its clock cycles to the end of the instruction. Registers are then saved and a jump done through a special ABORT vector at $00 FFE8 and $00 FFE9. This is important in a virtual memory system when a program attempts to address a range of locations which is not currently in RAM. The memory management can pull the ABORT line, get the new segment from disk on down into RAM, and then allow the program to continue.

The idea of a memory cache is foreign to microcomputer design, but it has the potential for yielding a spectacular increase in computational power. Most microcomputer memory management designs involve, for instance, a megabyte of RAM into which 16 megabytes of disk storage can be folded. In a cache system, there is yet another level interposed between main RAM and the processor, and that is a high speed RAM cache. Most RAM memory chips used in microcomputers have a cycle time of 200 nanoseconds. This limits useful microprocessor speed to about five MHz. To use a 10 MHz 68000, you have to use special 100 nanosecond RAM.

Current versions of the 65816 can run at 10 MHz and are fabricated with features about 4 1/2 microns (millionths of a meter) across. When you reduce feature size by half, you reduce parasitic capacitance (see Chapter 13) four times. The newest fabrication technology allows a reduction in feature size of about three times allowing for 1 1/2 micron features and thus an eight fold increase in speed. The result of all this should be an 80 MHz 65816. The problem with a beast of that speed is that standard, inexpensive RAM chips are hopelessly too slow to be of any use. There are very expensive one nanosecond RAMs based on some advanced transistor technology, but you would never contemplate a microcomputer with a megabyte of one nanosecond RAM due to the enormous cost. Enter the memory cache.

In a 65816 based cache design you could install 64K of 10 nanosecond RAM to serve as the program cache, and 64K of 10 nanosecond RAM to serve as a data cache. This would be backed up by a megabyte of 100 nanosecond main RAM and 16 megabytes or more of disk based storage. This requires a two tiered memory management system in which one tier sees that program and data segments are switched from main memory into high speed cache for use by the processor running at full 80 MHz speed. The second tier would serve the more standard task of moving segments from disk to main memory as needed. Powerful mini-mainframes such as the VAX 780 use a cache scheme in which they maintain a speed ration of eight to one between cache and main memory, but work with just 8K of cache memory with a two nanosecond cycle time.

The VPA and VDA pins are intended to support the cache/main memory tier of memory management. William Mensch's ambitious design for the 65816 anticipated sharp reductions in the cost of very high speed RAM so that the double 64K cache puts the system into the range of power of a high powered Amdahl, rather than a mid power VAX. The other element of Mensch's plan for a mainframe in an Apple is the need for large amounts of hard disk storage well in excess of the 15 or 20 megabytes often used for microcomputers. After three years of design work, Mensch was gratified that Maxtor announced a 600 megabyte hard disk for microcomputers the same week that the 65816 was announced.

Advanced Instruction Types

A few other advanced features on the chip include a COP instruction which is a special kind of interrupt for coordination with a floating point coprocessor and a Wait for Interupt (WAI) instruction which ensures that the 65816 will be ready to respond immediately when an anticipated interrupt arrives. Since this is a low power CMOS design where an inactive chip uses very little power, there is a Stop the Clock (STP) instruction which halts the microprocessor with all registers intact until a reset occurs. If the first instruction received is an Return from Interrupt (RTI), the 65816 will resume execution just after the STP instruction rather than doing a system initialization. Finally, there is a block move instruction which provides a very substantial increase in speed while moving ranges of data since there is no need for a move program to index the read and write addresses.

The 65816 is available in a 40 pin package in which the eight data lines double as the bank address outputs. The 65802 is completely plug compatible with the standard NMOS 6502 used in the Apple. Like the 65816, the 65802 has a 6502 emulation mode which gives it near total 6502 compatibility. With a 65802 plugged in, you cannot use the bank addresses, so you still have a 64K address range, but you can take advantage of such features as the 16 bit ALU, and the large number of new instructions.

There are 255 opcodes in the 65816 instruction set as opposed to 151 for the 6502, and these represent several new instruction types and 11 new addressing modes. Some of the new addressing modes pertain to the 24 bit addresses, so the useful set of 65802 instructions is not quite as large. You can write 65816 programs on your Apple with a 65802, but the highest address byte can't be announced onto the bus. Nonetheless, programmers who have worked with the 65802 at one MHz report speed increases of up to four times for tasks rewritten into 65802 code from 6502 code.

Some of the new instructions are also provided in the 65C02 (see Chapter 29), so they are supported by the ProDOS Assembler from Apple. However, the place to look for software support is Hayden, publishers of the widely respected ORCA/M assembler (see Chapter 31). Hayden was closely involved with the Western Design Center and has developed a number of system support tools such as the ORCA/HOST which supports all 65816 instructions, but which runs on the Apple II under ProDOS. Hayden is also developing a Pascal compiler which will produce either 6502 or 65816 code.

65816 Based Products

Aside from future plans to increase the speed of the 65816, the Western Design Center is planning a 32 bit version, tentatively called the 6581632, which would be mounted with a floating point processor and a local area network controller on board. Apple II designer Steve Wozniak is excited about the possiblity of designing systems around the 65816, but the exact

future of this and other projected WDC products is unclear at the time of this writing. Nonetheless, other Apple II owners may be enthusiastic about the future prospects resulting from what WDC designer William Mensch describes as his "mutual love affair" with the Apple II.

The first two products available for the Apple II with 65816 based designs are the Tool Box, which is a full scale development system for programmers, and a more expensive system for CAD microprocessor design and logic simulation. This design system is being used at WDC to replace a million dollar Prime computer logic simulator and a Kalmagraphics CAD design system. The 65816 and 65802 are currently manufactured by GTE under license from WDC.

The NS 16032

The 16032 family of microprocessors from National Semiconductor is yet another member of what may be considered a fifth generation of microprocessors. As you can see from Figure 30.4, it looks in some ways like a scaled down 68000, but the most important new feature is a 32 bit ALU. Like the 68000, the NS 16032 uses two byte instructions, but the 16032 instruction set takes better advantage of the 64,000 possible instructions to implement a variety of addressing and data structures which National Semiconductor claims will make it possible to greatly improve the efficiency of high level languages.

Another important new feature previously available only for minicomputers is the ability to move data directly between memory locations without first passing through a microprocessor register. This limits internal work to memory management and calculational tasks, and thereby provides a substantial reduction in processing time.

When a program written in BASIC, Pascal, or FORTRAN is compiled into machine language, the code which results tends to be comparatively sprawling and inefficient. This is in part because the objectives of microprocessor instructions with regard to data are usually quite different from those of the high level language. The NS 16032 instruction set is intended to provide machine language instructions which more closely reflect the kinds of tasks out of which high level programs are made. The hope is that compilers for the 16032 will produce code from high level language programs which is nearly as efficient as direct coding in machine language. This would remove all advantage of programming in machine language.

The 16032 is part of a family of chips the most elaborate of which is the 32132 which has a 32 bit external data path and a 29 bit address bus. The 16032 has a 16 bit data bus and a 24 bit address bus, like the 68000. National Semiconductor is also marketing a floating point math unit called the 16081 which can use a number with an 11 bit exponent and a 52 bit mantissa. This is therefore comparable to the 8087 from Intel. It can perform a 64 bit multiply in 6.2 microseconds. There is also a separate memory management chip called the 16082.

National Semiconductor's entries in the eight bit (SC/MP) and 16 bit (PACE/INS 8900) microprocessor market have not been extremely successful because of design problems. The company has therefore been focusing concerted attention on these 32 bit processors for some time and is well ahead of the other major microprocessor manufacturers in coming to market. There will be other important entries in this field, but the NS 16032 at least provides a glimpse of the kinds of machines that will be coming along during 1984 and 1985.

A single chip with a 32 bit ALU, 32 bit data path, memory management, and on board floating point capabilities would completely replace most minicomputers and would be a substantial threat to some current mainframes. IBM has been able to string together what amounts to three 68000s to emulate their 370 mainframe (IBM XT/370), and that sort of feat will be

increasingly more important as more 32 bit microprocessors such as the WDC 6581632, the Western Electric 32000 for AT&T machines, and the DEC MicroVax 2 make their way into microcomputer designs.

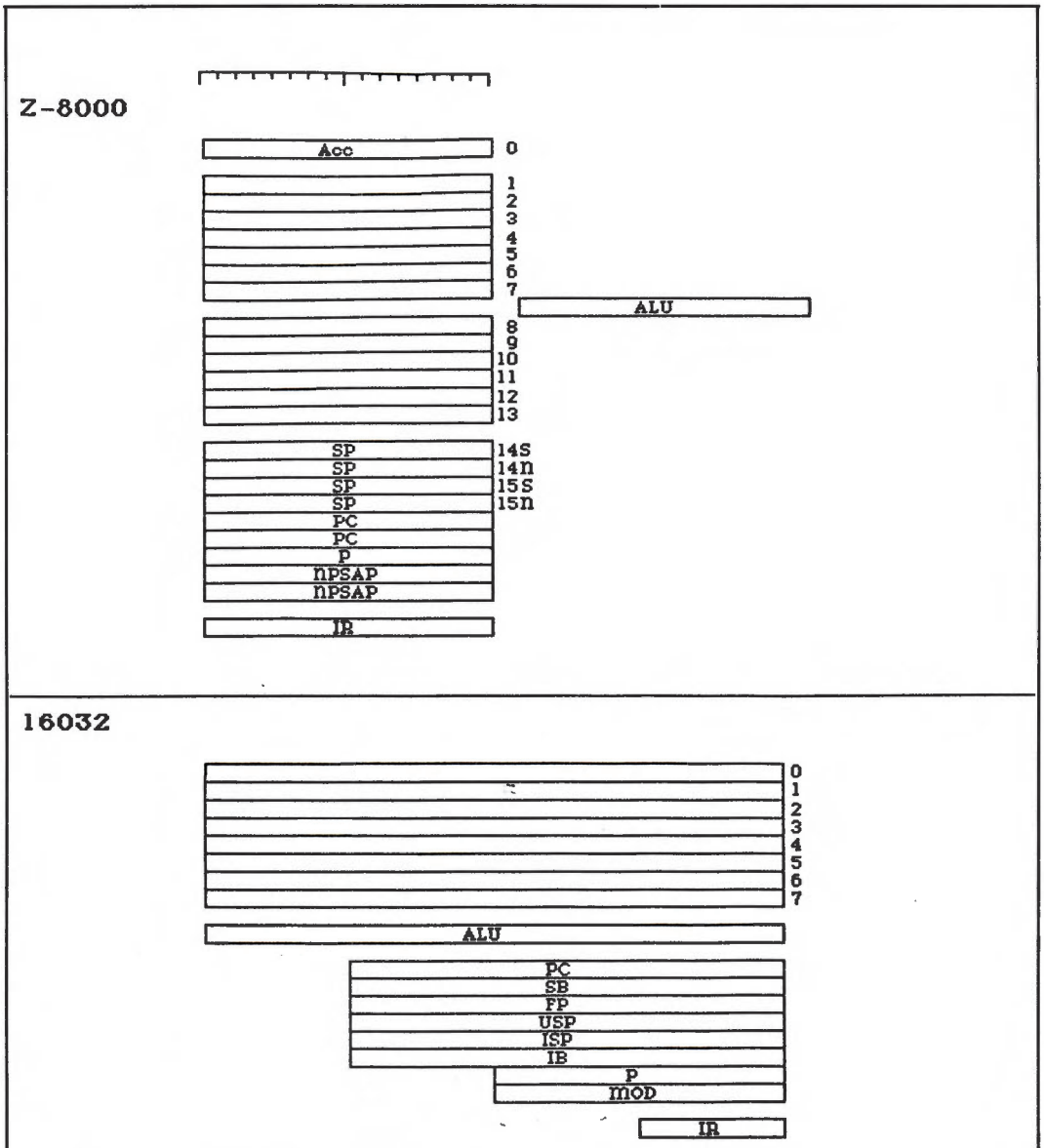


Fig. 30.4 Registers in Zilog Z-8000 and NS 16032.

Z-8000: Acc= accumulator, ALU = arithmetic and logic unit, SP= stack pointer, S= segmented, N= non-segmented, PC= program counter, P= processor status register, NPSAP = new program status area pointer.

16032: SB= static base, FP= frame pointer, USP = user stack pointer, ISP = interrupt stack pointer, IB= interrupt base, MOD= module.


SR
RL
RR
RLn
RRn
MASKL
MASKR
B
ASR
C
RLC
D
RRC
E
RLCn
F
RRCn
7
8
9
ABS
SHOW
SB
CB
GTO
LBL
HEX
DSZ
DEC
ISZ
OCT
BIN
4
5
6
CLEAR
SET COMPL
PRGM
REG
PREFIX
WINDOW
1'S
2'S
UNSGN
SST
BST
R↓
R↑
x≷y
PSE
BSP
CL X
ENTER
1
2
3
WSIZE
FLOAT
MEM
STATUS
EEX
f
g
STO
RCL
LST X
0
CHS
HEWLETT·PACKARD

Chapter 31

Assemblers and Debuggers

Creating Machine Language Programs for the 6502 or 65C02

Each of the instruction codes which are used to operate the 6502 can be represented in two different ways. One representation is intended to simplify the 6502's decoding task. In this form, each instruction is represented by a single byte "opcode" which can fit into the 6502's instruction register as a pattern of eight bits. For instance, the Jump To Subroutine command is sent to the 6502 as the bit pattern "0010 0000", which is equivalent to the hexadecimal number $20.

The other representation for the instruction code is intended as a compromise which can simplify the human programmer's task in writing lists of instruction codes yet still leave it interpretable for the machine. The human readable form of an instruction is called a mnemonic (the word root has to do with remembering things, and it is pronounced "nu mon ik" with the accent on the mon). For 6502 instructions, the mnemonics are always three letter codes such as JSR, which stands for Jump To Subroutine.

Machine Language, Assembly Language and Symbols

Lists of mnemonic human readable 6502 instructions are described as assembly language, while the same list written in purely machine readable opcodes is said to be in machine language. It is often convenient to write very short programs (up to 15 or 20 instructions) directly in machine language. For most programming tasks, however, it becomes so difficult to keep track of all the hexadecimal codes that the effort collapses once the program gets beyond that size.

Direct use of assembly language is also quite challenging because the program breaks everything down into a large number of very small but interleaved tasks. Although the instructions themselves are represented by mnemonics, all of the address and data bytes are still in hexadecimal. In fact, any instruction code program longer than 50 or 60 instructions has to be written in what is called symbolic assembly language. Here, descriptive "symbols" of up to eight characters in length replace the hexadecimal addresses.

The machine language statement:

```
$20 $78 $FB
```

becomes the assembly language statement:

```
JSR $FB78 (jump to the subroutine at $FB78),
```

but in symbolic assembly language the statement is written as:

```
JSR VIDWAIT (jump to the subroutine which outputs a
             character to the video screen and
             checks for a Control-S pause character)
```

Commented Source Code

This is still considered a bit difficult, so the tradition is to include detailed comments in the symbolic assembly language program. These "commented symbolic assembly language" programs are often called "source code." It is usually possible for someone with a little bit of introductory background in 6502 instruction code to pick up a copy of some source code and be able to figure out what is going on.

Apple provides complete source code for the Monitor ROM as part of the II/II+ reference manual and as part of the optional reference manual for the //e or //c. In the "source listing" for the Apple IIe, this line of code appears as follows:

```
JSR VIDWAIT     ;OUTPUT CHR & CHECK FOR CTRL-S
```

If you want to get more of a sense of these four forms of a 6502 instruction code sequence, you can try the following. Turn your Apple off and on, hit reset (CTRL-Reset) to stop the drive, and then type CALL -151 and hit Return to enter the Monitor. Now, type: FDF0.FDF7 and hit Return once and then, before typing anything else, type: FDF0L and hit Return once.

Reading Some Instructions

The first command you typed in caused a "hex dump" of eight bytes of a program stored in the Apple's memory. Immediately below it you should also have caused an assembly language listing of the same instruction sequence. You can match up the C9 A0 as the first few bytes of both listings (2C 7B 06 are the first three bytes in the //c). In the assembly language listing the hex codes are listed one, two or three at a time, adjacent to their assembly language interpretation.

This happens to be the beginning of the COUT1 routine which sends characters to display memory for output by the video system (see Chapter 33). The name COUT1 is itself a symbolic label which identifies the starting point of the subroutine for character output. Program flow is a little different here in the //c, so //c owners should now type: FBB6L, followed by a Return before reading on.

The first instruction (CMP #$A0) checks to see if a character headed for the screen is actually a control character (all control characters have Apple ASCII codes less than $A0). The next

instruction line has a conditional branch instruction (see Chapter 27) with a relative address. The "90" is equivalent to the Branch if Carry Clear (BCC) instruction, and the "02" tells the program counter to advance by two.

If the incoming character is a control character, the 6502 will take the branch and skip two locations to $FDF6 instead of continuing immediately at $FDF4. In the //c, the branch causes a skip to a JMP $FDF6, which gets it back in line with the older II+ and //e systems. The instruction being skipped is the AND with the "inverse flag" (see Chapter 33) which controls whether or not the character should be displayed on the screen or in inverse. Control characters aren't supposed to go through the AND operation and the BCC takes care of this.

Now that you have a general sense of all this, you can look on page 150 of the *Apple II+ Reference Manual* and see the symbolic listing or on page 15 of the *//e Reference Manual Addendum* and see the fully commented source code.

Assemblers Turn Source Code into Machine Language

As explained, if you're interested in learning to create your own machine language programs, your task will be infinitely easier if you actually write in commented symbolic assembly language (source code). However, if you do write this kind of source code, you must also have access to a program called an Assembler which can translate your source code into machine language "object code" once the program is ready to be run.

An assembler program is the same sort of thing as the Applesoft Interpreter or a FORTRAN Compiler. You enter "verbose" commands, and the assembler turns your human readable program into binary machine language. To work with 6502 machine language, you must learn the details of the 6502's instruction codes, learn the layout of the Apple's address space, and choose an assembler program and learn to use it.

Books on 6502 Instructions and Apple Organization

This is another one of those situations where owning an Apple puts you in a very good position. Unlike virtually any other microcomputer system, the Apple has been the focus of interest for huge numbers of beginning machine language programmers. There are nearly a dozen good books on 6502 machine language, several of which have sold more than 100,000 copies, so you are not alone.

The two best books for beginners are *Assembly Lines: The Book* by Roger Wagner (Softalk Publishing), which is a collection of articles from *Softalk* magazine, but which also includes a complete review of the 6502's instruction set. Also, a book by Randy Hyde called *Using 6502 Assembly Language* (DATAMOST), which is a more thorough treatment, but also directed at beginners. More advanced information is included in *Programming the 6502* by Rodney Zaks (Sybex), and you can work your way up to professional level with *6502 Assembly Language Programming* by Lance Leventhal and *6502 Assembly Language Subroutines* by Lance Leventhal and Winthrop Saville (both from Osborne/McGraw Hill).

If you are a //c owner you may want to stick to 6502 code so that your programs will run on other Apples. If you want to learn more about the 6502, however, you'll have to watch carefully for new editions of these books.

Apple Thesaurus is intended to provide a general overview of what is out there in the address space and a sense of how the whole thing works as a system. Two more detailed books are *What's Where in the Apple* by William F. Luebbert (Micro Ink) and *Apple II Monitors Peeled* from Apple. It is also essential that you have a copy of the *Apple II Reference Manual* for your machine. This was included with the Apple II and II+, but if you are a //e or //c owner, you will have to ask your dealer to get ahold of one for you. The //e or //c manual should be accompanied by a separate addendum which includes the commented source code listing for the //e monitor and 80 column card.

Choosing an Assembler

As a consequence of the enormous interest in dabbling in 6502 instruction coding, there are quite a few assembler programs available out there. These programs vary enormously in power and in intended audience.

Everyone who has an Apple II with at least 64K of RAM, and those few who have the old Integer BASIC ROM, can use what Apple calls its Mini-Assembler. This is not a symbolic assembler, so it is not useful for a program much bigger than 20 or 30 instructions and is not too good an idea, even for learning with. Once you have a little skill, it is very convenient for quick off the cuff assembly of short subroutines. Its lasting popularity among progammers has finally caused Apple to put the Mini-Assembler back into the revised ROM for the //e, which became available in the summer of 1984.

Everyone else can get the Mini-Assembler running by booting a DOS 3.3 master in a 64K (the minimum) Apple. The DOS master loads Integer into the language card. Which means that a complete copy of the original Apple II ROM is being loaded into the high 16K of RAM. Next, you type INT and hit Return—this causes bank switching which hands control to the old Monitor. Now type CALL-151, and finally, F666G. Apple also sells the 6502 Assembler/Editor with the Applesoft Toolkit, but if you're going to buy an assembler, you can probably do a lot better with one of the other offerings.

If you buy ProDOS, however, or get it included with your original Apple purchase, then you can use one of the most powerful and diverse assemblers available for the 6502, the ProDOS Toolkit Assembler. Other powerful assemblers include Merlin from Southwestern Data Systems, LISA (pronounced Ly Za) from On Line Systems, ALDS (Assembly Language Development System) from Microsoft, and ORCA/M (Object Relocatable Code Assembler/ for Microcomputers) from Hayden Software.

Each of these has special strengths with ORCA/M, ALDS, and the ProDOS Toolkit being best for very advanced work, and with Merlin and LISA being good choices for beginners. The ProDOS Toolkit Assembler is very good for beginners as well as for professionals, so it will probably be the first choice for those who don't need to work with DOS 3.3 or in Z-80 code. //c owners should try to get a version of one of these that will work with the full 65C02 instruction set.

Typing in the Source File

In order for an assembler program to translate your verbose, commented source listing into a precisely determined machine language object program, you must give the assembler a lot of cues. One of the important things is the precise layout of the labels, mnemonics, "operands"

and comments in your source file. Everything is required to be in neat columns, all in register and with no exceptions.

Advanced programmers are expected to be able to just load up their word processor and go about creating the file. You must be careful of the line numbers, tab positions, etc. Of the various assemblers mentioned, only ALDS gives you no help at all in this task. ProDOS, ORCA/M, Merlin and LISA all provide an editor which is distinguished by providing what are called "dynamic" or "relative" line numbers as you work, and by the fact that they automatically format the source file with a minimum of cues from you as you work.

Line numbers in an assembly language source file are different from those used in a BASIC program in that they must be in an uninterrupted sequence from one to the highest line number with each new line being just one higher than the preceeding one. These editors handle this automatically, readjusting all the line numbers every time you add or delete a line.

In addition to formatting the file into clearly separated label, mnemonic, operand, and comment fields, all of these assembler editors provide for some degree of word processor-like editing. Most are simply "line editors" in which you identify the line you want and then make changes. The ProDOS Toolkit and ORCA/M provide "global search and replace," and the ProDOS system actually lets you edit two files at the same time which is very handy for some advanced programming problems.

The chief claim to fame of On-Line's LISA is that it is actually an assembly language interpreter rather than a compiler like the others. This gives it the feel of entering an Applesoft BASIC program. As each line is entered, it is tokenized and examined for syntax errors. This makes it a wonderful assemblers for beginners. Unfortunately, LISA doesn't do too well with advanced programming tasks, and worse, the designers of LISA decided to improve on some otherwise standard cues to the assembler. This means that if you try to enter a source listing from some magazine, LISA often can't assemble it until you change the cues to its own version. This is rather rough on beginners.

Pseudo Opcodes to Guide the Assembler

In addition to labels and mnemonics, it is possible to include special instructions to the assembler on exactly how it is to proceed. For instance, you can let it know that you're going to be entering a string of ASCII codes which it shouldn't try to translate into anything. These internal instructions are called "pseudo opcodes" because they are typed in to look like three letter mnemonics, but do not get turned into 6502 opcodes. The assembler assimilates the information as it works and then removes them completely. These pseudo opcodes can be used to tell the assembler about the actual position in memory where a program is going to be placed when it is run, how it should be typed out on paper, etc.

The various assemblers vary in the variety of their pseudo opcode dialects. ALDS provides only about eight or 10 very simple pseudo opcodes, while the ProDOS Toolkit assembler has 40 of them. In addition, there are some special instructions called "Sweet 16" opcodes which permit an assembly language program to take advantage of some 16 bit math routines in the Integer BASIC ROM. These are not always convenient to use, but Merlin, LISA and the ProDOS Toolkit all provide facilities for using them if you want to.

Using Macros to Simplify Programming

Most of the high speed math, sorting, and graphics routines that you could ever possibly need have been written again and again and again, and, in some cases, they have been written by programmers with substantial skill. The ideal programming environment lets you work with a library of these assembly language routines as well as a collection of your own routines. As you write a program, whenever one of these routines is needed, you simply put a notation in your source file. Later when the assembler goes to put the whole thing together, calculate all the addresses to replace the labels, etc., it pauses at each of your notations, goes out to disk to grab the source code for that routine from its library, inserts it, and assembles it into the final machine language listing.

These inserted routines are called "macros," and it is best if you have an assembler which can handle macros. The task is not trivial, because it involves substantial complexity in making the address references tie in correctly. For example, if your current program references some location in the macro, the assembler can't assign a numeric value to that reference until it first inserts and evaluates the macro, but the macro may refer back to the section of your program that called it, etc. If all of this isn't handled just right you get tangled spaghetti.

Merlin provides modest macro facilities and a library system, but it is limited in the complexity of "nesting" of macros one within the other, and in the complexity of the relationships among interacting variables. The ProDOS Toolkit and ORCA/M have very elaborate macro facilities and should meet the needs of most 6502 programmers. The most elaborate macro and link facilities are provided by ALDS, and the ALDS macro library is phenomenal.

The warning on ALDS is that it is not for beginners. It actually runs on the Microsoft Z-80 card, and can work with 6502, 8080 or Z-80 code and can create programs which call back and forth from the 6502 to the Z-80. Microsoft has provided a library of the Z-80 routines from their own versions of FORTRAN and COBOL which can be called from within a 6502 program.

Relocation to Make the Programs Versatile

Looking back up to the example lines toward the beginning of this chapter, you will see that the principal difference between a symbolic assembly language program and a simple mnemonic and operand listing is that the labels have been replaced with absolute addresses, VIDWAIT becomes $FB78. If Apple ever had to add a few extra lines and move the whole program down in memory by a few locations, then every address in the program would become invalid. With a large program, there is no choice but to go back to the symbolic source code and reassemble the whole thing at its new location, generating a new set of addresses for the whole program. This process is called relocation.

For smaller programs it is convenient to be able to create a version which is mostly assembled, but in which the actual addresses for each label have not yet been inserted into the program. These partially assembled programs are sometimes called relocatable files, and the program which finishes the assembly as the program gets loaded into its actual destination is called a Relocating Loader. Only ORCA/M and the ProDOS Toolkit Assembler have substantial relocation capabilities.

Supplementary Services from Assemblers

There are several things which some assemblers do that can be very helpful in programming although not essential. Both Merlin and the ProDOS Toolkit can generate "symbol tables" which help you find your way around your own program. These will list all the symbols in alphabetical or in numeric order, and show where they are used within the program.

Another important extra is "disassembly." This is the ability to read in a machine language file and generate an assembly language file. This is available with LISA and with Merlin, which are symbolic disassemblers. This means that they will replace addresses with alphanumeric notations. This facility can be used to relocate programs, but Merlin does something even better. It searches its internal listings to see if any of the addresses refer to known subroutines in the Apple Monitor or in Applesoft, and puts in the accepted label names (such as putting in COUT1 when it encounters $FDF0).

The most wonderful thing that Merlin does, and well worth the price of the package for determined programmers, is that it can generate a fully commented source listing of the Applesoft Interpreter. The labels and comments are stored on the Merlin disk. This is something Apple will not release, and there is no guarantee for programmers that addresses will stay stable on newer Apples, but since it's based on the ROM in your own machine, it ought to be good at least for home fiddling.

Debuggers for Machine Language Programs

Once your program is written, assembled, loaded and ready to run, you're probably not home free just yet. Much like in BASIC, programs just don't work the first time you try them. Structural and syntactical errors usually get picked up during the assembly process. What remain are all those dozens of contingencies you didn't quite think of when you were writing.

For major overhauls, you go back to your source code and rewrite with the editor. However, the whole process of changing source code and reassembling is lengthy, and as the problems get more subtle, you typically need to work at the machine language or non-symbolic assembly language level to identify and correct problems. All microcomputer systems include a "debugging" tool for working in this fashion. The Apple's "Monitor Command Processor" (usually just called the Monitor since MCP never seemed to catch on) is such a debugging tool.

When working in CP/M with a Z-80 or 8080, you use a bug killer program called Dynamic Debugging Tool (DDT), and in 68000 systems you can use either MACSBUG or a special system monitor provided by your 68000 company's programmers. The old Apple Monitor is a bit weak compared to DDT, and MACSBUG is better still. If you buy ALDS for your Z-80 card, you can also get DDT65 which is a 6502 debugger that also uses subroutines running on the Z-80.

Once again, there's good news for ProDOS users. The ProDOS Toolkit includes Bugbyter which is far and away the most powerful machine language debugging program available for any of these microprocessors.

Features of a Debugger

The essential features provided by the Apple Monitor Command Interpreter include the ability to examine hex dumps of memory and to produce a non-symbolic disassembly (see Fig. 31.1). This lets you examine the actual contents of your machine language program, or see it listed in mnemonic form. You can use the Monitor to change hexadecimal values and to copy your program from one place in memory to another. Once you've made your changes you can give the machine language equivalent of a RUN command.

In addition, the Apple Monitor permits you to examine and change the contents of the 6502's internal registers. Other miscellaneous services include doing simple hexadecimal arithmetic, changing the display to inverse, and saving your program out onto a cassette tape.

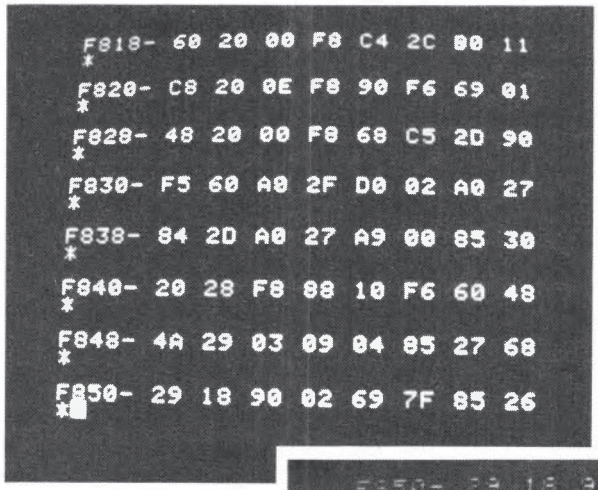


Fig. 31.1a Hex dump listing of memory between $F818 and $F857.

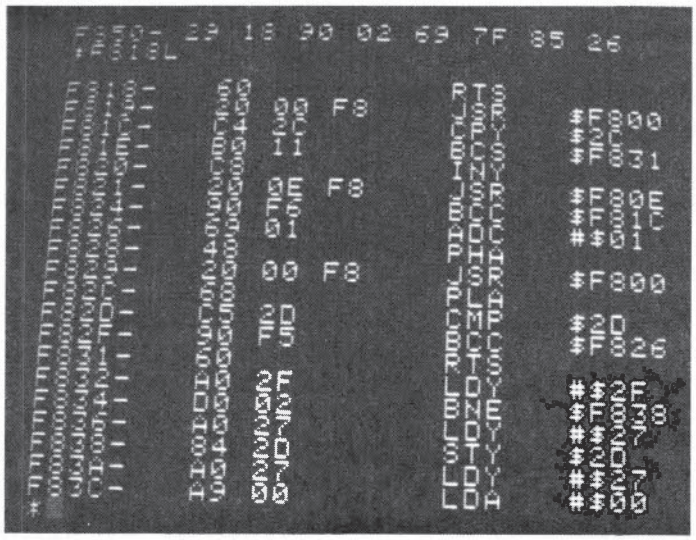


Fig. 31.1b Disassembled listing of memory between $F818 and $F83C.

Shortcomings of the Apple Monitor

This is just enough to operate effectively, although in DOS 3.3 you do have to come back up out of the monitor to move your program on and off the disk drive. Other than the lack of disk commands, there are three glaring weaknesses in the Monitor. First, there is no way to get an ASCII dump in which ASCII codes stored as hexadecimal numbers in memory are displayed as the letters they are meant to represent. Second, there is no facility for step and trace, which lets you execute your machine language program one instruction at a time in order to follow all changes in registers and memory locations. Third, without the Mini-Assember you cannot conveniently assemble short lists of mnemonics into machine code on an active basis while you are in there working.

All of these features are available in DDT for the Z-80, DDT86 for the 8088/8086, DDT65 for the 6502 with a Softcard, and in MACSBUG. In addition, MACSBUG for the 68000 permits the programmer to compute complex indexed addresses, to do Hex/Dec and Dec/Hex conversions, and it provides an enhanced kind of step and trace with transparent break points which lets your program run at full speed up to a predetermined point and then halt, but in which you don't actually have to insert break commands.

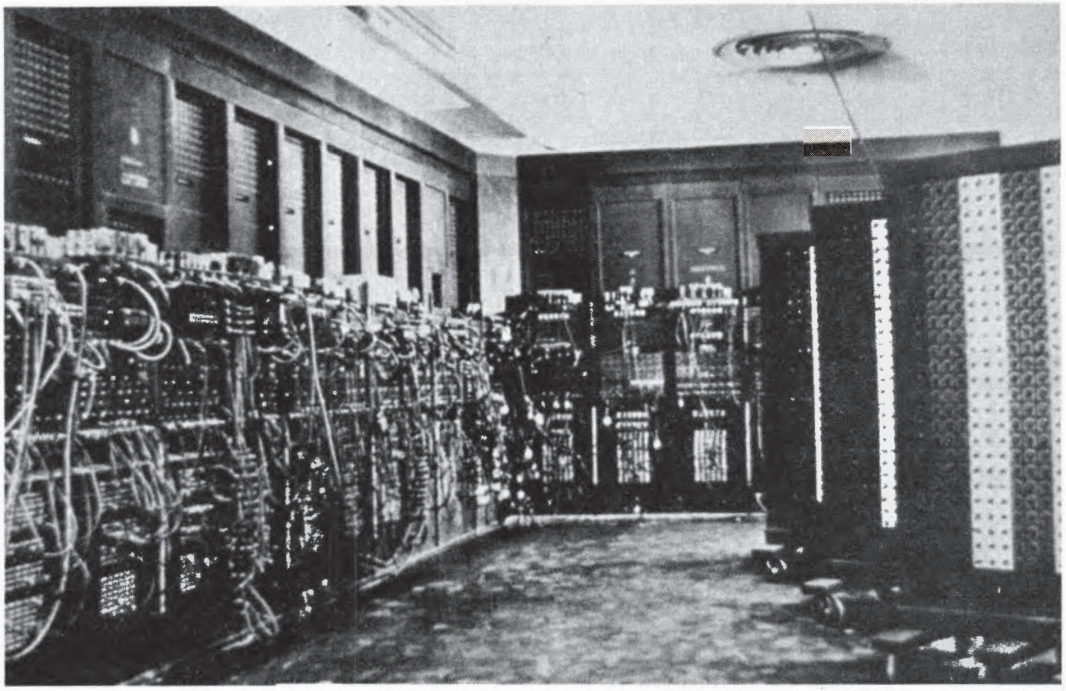
The Full Featured ProDOS Bugbyter

The Bugbyter for the 6502 with ProDOS lets you use all of the Monitor commands and provides a variety of very powerful enhancements. It permits you do issue all ProDOS disk commands directly from within Bugbyter, does ASCII dumps, and duplicates the powerful transparent break point system of MACSBUG. But its real power is in its innovative and striking step and trace system.

You can actually connect a joystick and control the rate of program execution, and, as the program runs, you can choose to see an active display of the changing microprocessor registers, changes in a chosen area of memory, or changes in the stack. All this is in real time as your program executes under the speed control of your joystick. For assembly language programmers who have toiled for years without such wonders, it is a delightful and entertaining experience; for beginners it is enormously helpful; and for folks interested in only a reading knowledge of assembly language, it is sort of fun to watch and play with.

PART 3

The Processing Apple



Chapter 32

Operating System Structure

In the beginning, there was only hardware. Then there were a few programs and a few programmers. Finally, a day came when two programmers wanted to use the same machine at the same time. They put a calendar on the wall and set up a sign-up list. And so the first operating system was born. This primal operating system function is called "scheduling CPU time."

The next great advance in Operating System (OS) history was the creation of a program that could make the wall calendar sign-up list obsolete. Yes, it was an automatic job sequencing program called the "Monitor." The Monitor was loaded into the computer's memory and stayed there at all times. You were no longer allowed to just start using the computer, you had to get your program ready to go, and then ask the Monitor if it could see about running your work through the CPU.

Not long after the creation of the Monitor, a program accidentally overwrote and destroyed the Monitor. The founders responded to this with the invention of Memory Management. This was a way of reserving particular areas of computer memory for particular uses. With Memory Management in place, both the Monitor and the user's program could be in computer memory at the same time without stepping on each other.

Soon, everything was running smoothly in a way any computer would approve of. Early in the morning, dozens of humans would arrive with stacks of punched cards to dump into the card reader. The Monitor would read through the cards as it saw fit. During all this, the humans waited patiently in another room and in no way hampered the speed of the computer's operation. The slowest part of the system was often the mechanical card reader. Aha—the next target for computerization was therefore: selection, operation, and buffering of I/O Devices.

The Monitor was therefore expanded to include a collection of I/O Subroutines. With the addition of different makes of card readers, and then magnetic tape drives, the Monitor was set up with a collection of subroutines it could call and run when it wanted to operate these different Input/Output (I/O) devices. Further, any running program could issue requests to the Monitor for the use of its I/O Services. The full set of services would include activating the device, moving the data and giving a full report to the program that requested the service.

This was all so fast and convenient that the amount of data kept in external mass storage grew even faster than the number of users. Independent of the needs of operating the I/O devices, there had to be a system for keeping track of what was stored where by whom. Thus the Monitor was expanded once again to include a File Management system.

With the grand master Monitor supervising CPU Scheduling, Memory Management, I/O Services and File Management, tens of thousands and then hundreds of thousands of computers poured forth into the world. The enormously increased base of users tended to respond to all this by thinking that it was all well and good that the devices worked so well and had such nicely organized management, but if these things were such a marvel of coherent planning, why was it so hard to communicate with them? And so was born the friendly, interactive User Interface.

Scheduling the Processor

DOS on the 6502, CP/M on the Z-80, and MS-DOS on the 8088 have fairly simple approaches to supervising the microprocessor. Only one program is running at a time, so all the operating system has to do is to serve as a sort of background program which is active when nothing else is going on. When a user program needs to be run, the operating system keeps track of the entry point to that program and gets things running by doing an effective JSR to that entry point. When the user program finishes, control comes back to the operating system's background level.

Things can get much more elaborate, however. Digital Research offers an operating system called MP/M which allows several users to work simultaneously on a single Z-80. MP/M must supervise the various users and shift the microprocessor's attention from task to task constantly, allocating approximately equal amounts of time for each. Multi-user microcomputers provide a modest improvement in cost, but performance drops off since one Z-80 running three programs "simultaneously" does each of them at a third of its normal speed. Things are a little better with an 8086 running MP/M 86, but most newer multiuser systems have relied on the more powerful Z8000 (in the Onyx) and in 68000 based systems which can run UNIX.

Multi-Tasking

Much more important in microcomputers is "multi-tasking," the ability to have several programs active at the same time in one microcomputer being used by one person. In these systems, the operating system lets the user assign priorities to the various tasks and lets the user select which one of the programs will be responding to the user's actions at the console. The various programs run concurrently, in that they are all at least partially in memory and each get some of the microprocessor's attention.

These systems make it easy to shift from one task to another at whim, much as folks are accustomed to doing in their non-computer work. All that is important is that the various programs and their data all be in memory and have their own file buffers for disk access. The recent trend is to reflect all this on the video screen with multiple "windows," each reflecting the activities in one of the optional tasks (see Chapter 37). The better systems can actually make very efficient use of a microprocessor by, for instance, doing calculations for one program while waiting for keypresses in another. This kind of active concurrency depends on a timed interrupt system which lets the Monitor regain control periodically and make decisions about which tasks should be carried out in the interval before the next interrupt.

A modest multi-tasking capacity is available on the Apple III, and ProDOS includes some of the necessary facilities for doing this on the Apple II if the mouse interface is installed. Users of the Rana 8086/2 (see Chapter 30) can get Microsoft "Windows" and Concurrent CP/M 86, which also provide modest multi-tasking abilities. 68000 based systems such as the Lisa or the IBM PC 3270 have much better performance in that there is provision for integration among the several different programs.

Apple II owners can also do very advanced multi-tasking work by installing a $1,500 68000 board from Saybrook which can run UNIX Version 7. UNIX is a programmer's operating system not well suited for the applications user, but which has powerful processor allocation management. One popular feature of UNIX is the option to use "pipes," which means letting two or more programs interact with each other while they are all running on the same processor.

For instance, one program could be performing numerical analysis while the second program accepts the results of each calculation for effectively simultaneous statistical work. UNIX has a long tradition of academic and scientific use in minicomputers, and while $1,500 is a lot for an Apple peripheral, this is a tenth of the cost of a standard UNIX system and it can equal the performance of the older systems quite admirably. Putting UNIX in an Apple also makes sense because the same computer can be used for a variety of popular microcomputer applications on the 6502, a Z-80, or even on the same 68000.

Other than UNIX, it is quite possible that other, more "user friendly" multi-tasking operating systems will appear for Apple II 68000 boards. This is an option which Apple itself is giving close attention because of the potential for compatibility with Lisa and Macintosh.

Memory Management

The two principal areas of concern for Memory Management are taking care of what program uses which RAM when, and simulating the existence of RAM that doesn't exist via a technique called "virtual memory management." The Memory Management services available to the DOS 3.3/Applesoft programmer are fair to middling. They work, but are not particularly flexible or efficient. The details of memory management for Applesoft programs are reviewed in detail in Chapter 40.

The Apple //e and //c $C300 ROM contains routines (AUXMOVE at $C311 and XFER at $C314) for copying data between the auxiliary and main memory and for transferring program control to the auxiliary bank (see page 77 in the Apple //e Reference Manual), but these hardware control routines are not integrated into a full memory management system. ProDOS contains some routines (at $BFA0) for managing switching of the high 16K (see Chapter 25), and it also implements a bit map which can be used to mark any area of free memory as reserved. This is an important capability for the future implementation of any multi-tasking with ProDOS.

For the most part, the routines which manage bank switching of up to a megabyte of RAM into the Apple's memory space are provided by vendors of individual memory cards (Legend, Saturn, Abacus) or by companies such as Microsolutions, whose software lets VisiCalc models extend over large ranges of memory. The MagiCalc program from Artsci automatically detects and uses up to 512K of memory, and there is no reason why these features could not be incorporated into future versions of ProDOS.

Protecting Memory Segments with an 8088

The first wave of really elaborate memory management systems for microcomputers was kicked off by the window system of the Apple Lisa with its one megabyte of RAM, and has percolated down to the IBM PC environment for users with MS-DOS 2.0, usually for 256K of RAM. All of this is available to Apple owners by getting the Rana 8086/2 system which runs the Microsoft Windows operating environment.

Memory management is fairly straightforward in an 8088 or 8086 based system because the microprocessor views memory as a series of four distinct 64K clumps. While a program is running, the 8088 is generally unable to cross a boundary from one 64K region to another unless a program specifically alters its "segment registers" (see Chapter 30). In this fashion, several different programs can each be loaded into their own 64K segment at various places in the one megabyte address range. The Monitor part of the operating system can effectively switch in one program segment or another by altering the segment register, and this simultaneously protects the other segments since they become inaccessible to the microprocessor.

The Banked Version of CP/M Plus

Digital Research has released a new version of CP/M for the Z-80 called CP/M Plus (or CP/M 3.0) which has some memory management capabilities. With the banked version of CP/M Plus installed, the high 16K of memory is treated as common space which is never bank switched out of the main address space. This 16K includes some segments of the operating system, some buffers, and some space assigned to the running program. In normal operation while a program is running, this 16K common bank sits above "Bank 1" which contains the remainder of the active program (Transient Program Area or TPA). An additional 48K range called "Bank 0" can be loaded with the remainder of the operating system as well as any additional resident system extensions added by a systems or applications programmer.

When full operating system services are needed, bank 0 is switched into the address space and most of the running program becomes inaccessible. However, by storing an elaborate set of system services in the 48K of RAM in bank 0, CP/M Plus avoids disk accesses. The user experiences the benefits of a large RAM resident operating system without giving up space for the running program in bank 1.

Up to 14 additional banks can be used as buffers for data moving between RAM and the disk drives. Whenever a disk read is performed, the data can be brought first into one of the extra banks and then copied into bank 1 for use. With each read, more and more data accumulates in the supplemental banks. Later, when some data from disk needs to be used again, it may be already available in one of the banks, so no slow disk read is required the second, third, etc., time around. This version of CP/M 3.0 is available to run on the Gold Card from Digital Research.

The CP/M Plus banked system greatly enhances the performance of disk intensive software such as WordStar and dBase II, particularly if you are working with very large files on a hard disk or high capacity floppy, but it does not allow the loaded running program itself to be any larger than what can fit into bank 1 and part of the common area. Further, CP/M Plus loses track of what it has put where when you do a warm start (CTRL-C), so the banks have to be reloaded after disk changes.

On Chip Virtual Memory Management

The trend with more advanced microprocessors (80286, 68020, 65816) is to include virtual memory management capabilities on chip with the microprocessor. The idea is to write programs as if there were 16 megabytes of RAM available but to run them with just one megabyte actually installed. The installed RAM is broken up into logical pages or segments whose effective addresses can be altered. The operating system loads some of the program and data into RAM, and processing begins.

If the microprocessor generates an address for some program or data that is still on disk, the system must halt the microprocessor, copy the information into an area in RAM and then adjust the address of that RAM so it will respond to the microprocessor's request. Once the new information is loaded and the RAM addresses have been reconfigured, the microprocessor is allowed to continue processing.

The microprocessor or system hardware has to take care of most of this work, but the operating system must still be responsible for keeping track of which areas of RAM it is safest to overwrite, as well as which pages of memory are still on disk and which are currently loaded. There is usually some sort of optimization scheme to avoid overwriting program or data which will be needed again soon.

I/O Services

The I/O Services must provide for calling hardware control routines for character oriented I/O (keyboards, CRT terminals, printers and modems), sector oriented I/O (floppies and hard disks), date and time checking, bit mapped graphics, A/D and D/A (analog to digital and vice versa) conversion, and interrupts from peripherals.

The chapter on Apple Video provides substantial detail about the lowest level of the bit graphics system. The actual calculation of dot patterns is done by the hi res routines in the Applesoft BASIC Interpreters (see Chapter 38). Apple has a long tradition of building good bit mapped graphics into its machines, while most CP/M systems never offered graphics. Digital Research now offers a graphics extension to CP/M which provides management routines, but the hardware control routines must be implemented by the individual computer manufacturers. Most non-Apple CP/M systems use character oriented CRT terminals for display, so bit mapped graphics just isn't an option.

Details of clocks, A/D conversion, and interrupts are covered in Chapters 9, 14 and 15, and sector I/O at the level of hardware control routines is discussed in Chapter 23. More details on sector I/O are covered along with the discussion of File Management in Chapters 34 and 35.

The inner workings of the Apple's character oriented I/O system gives considerable insight into the operation of the Monitor Command Processor, the Applesoft BASIC Interpreter, and some aspects of DOS and ProDOS, so it is covered in substantial detail here and in Chapter 33. These character I/O routines are stored in ROM on the Apple's motherboard.

Hardware Control Routines

An important example of an I/O Service is the Apple's GETLN routine. When a program wants to collect a full line of new data from the person at the keyboard, that program can simply call for the GETLN system service to go into action. GETLN's fundamental purpose is to "call" the keyboard input routine, but its management role also involves a variety of additional support features.

GETLN will put a prompt and a cursor on the video screen to let the operator know the keyboard is being watched, and then it will call the actual keypress input routine repeatedly until the person hits a Return at the end of a completed line. It also provides facilities for letting the person do typing corrections on the line before hitting Return, calling video control routines to display the typed characters, and depositing the new line of data in an agreed upon place within the computer's RAM for the original calling program to use.

Therefore, the simple assembly language instruction "JSR $FD67," which calls GETLN, is a very powerful and economical way of handling a complex but highly standardized input/output task. This management routine ties together groups of hardware control subroutines to provide a unified, but more complex, I/O service.

Device Independence and I/O Hooks

Apple DOS 3.3 is famous for the comparatively effortless control it provides over hundreds of kinds of character oriented input and output devices and this feature has been made even more versatile in ProDOS. The key to this device independence is the input/output hook system which is based on two vectors in the Apple's zero page called CSW and KSW (see Chapter 33). Among other things, these two vectors hook together the supervisory level of the GETLN I/O service with the actual hardware control routines for the video system and the keyboard. CSW contains the starting address of the hardware control routine for an output device, KSW serves the same function for input. GETLN doesn't care what kind of equipment those routines operate, as long as the routines respond to the conventions of GETLN (see Chapter 33, Figure 33.1).

It is these two locations you access when you type PR# or IN#. In DOS, you can set them to point to hardware routines on cards in slots one through seven (a PR#4 loads CSW with $C400—see Chapter 22), and in ProDOS, you can set them to point anywhere you want in the 6502 address space (i.e., PR# A$0303). As you will see in the discussion of the DOS and ProDOS command interpreters (see Chapters 34 and 37), there is an extra wrinkle before CSW and KSW get set, but that will make more sense when their standard function is explained.

Nearly all manufacturers of Apple peripherals provide hardware control routines in ROM chips on their peripheral cards, all of which are written to fit smoothly into the Apple operating environment's CSW and KSW system. This is why you can choose from among hundreds of kinds of hardware and thousands of kinds of software, and still have a reasonable chance of getting everything to work together. The programs can call on the Apple's system services, and those services can operate any hardware whose control routines follow Apple's system convention.


ROYAL

Chapter 33

Apple Character I/O with GETLN

The GETLN Monitor ROM Subroutines for Character I/O

Most Apples in the world spend 99 percent of their time running around and around and around in a very small programming loop inside the GETLN routine which loop is called KEYIN (or GETKEY for //e 80 column users). If you are running DOS or ProDOS and there is a cursor on the screen, then your Apple is probably trapped inside KEYIN (flashing cursor) or GETKEY (solid cursor) at this very moment.

To let your Apple break out of KEYIN (or GETKEY) all you have to do is push down a key on the keyboard. If you pushed the Escape key, the Apple goes into the RDCHAR routine for a moment and then plunges right back into KEYIN (or GETKEY) to see what kind of escape sequence it is supposed to carry out.

Most key presses, however, get you all the way out of RDCHAR into NXTCHAR. NXTCHAR is a routine which does a variety of things, but whose two chief claims to fame are: storing the character you just typed into the Apple's Input Line Buffer where any program can easily find it, and sending the character out to the video system so you can see what you just typed.

NXTCHAR then points the 6502 back into KEYIN (or GETKEY) again and again until you have either typed in 256 characters or hit Return at which point it guides the 6502 back up out of the GETLN area back to the running program. Often, the running program grabs the line of input out of the Input Line Buffer, stores it somewhere else or analyzes it as a command, and, when it's ready to proceed, calls GETLN again and dumps the 6502 back into the KEYIN (or GETKEY) loop to wait for a keypress (see Figure 33.1).

Whenever a character needs to be sent to the video screen, some part of the GETLN system will call for the COUT system to go into action. In the Apple II/II+, this usually means an immediate entry to the COUT1 routine, while in //es with the 80 column firmware on, it leads to the equivalent routine BOUT (after making a few stops along the way). These COUT routines take care of putting each character in the correct position on the screen, and do scrolling whenever necessary. They also watch for any Control characters which call for special action with regard to the text display. (The //c system programmers had to economize on ROM space, so they actually use entries through KEYIN and COUT1 rather than special new entries as in the //e.)

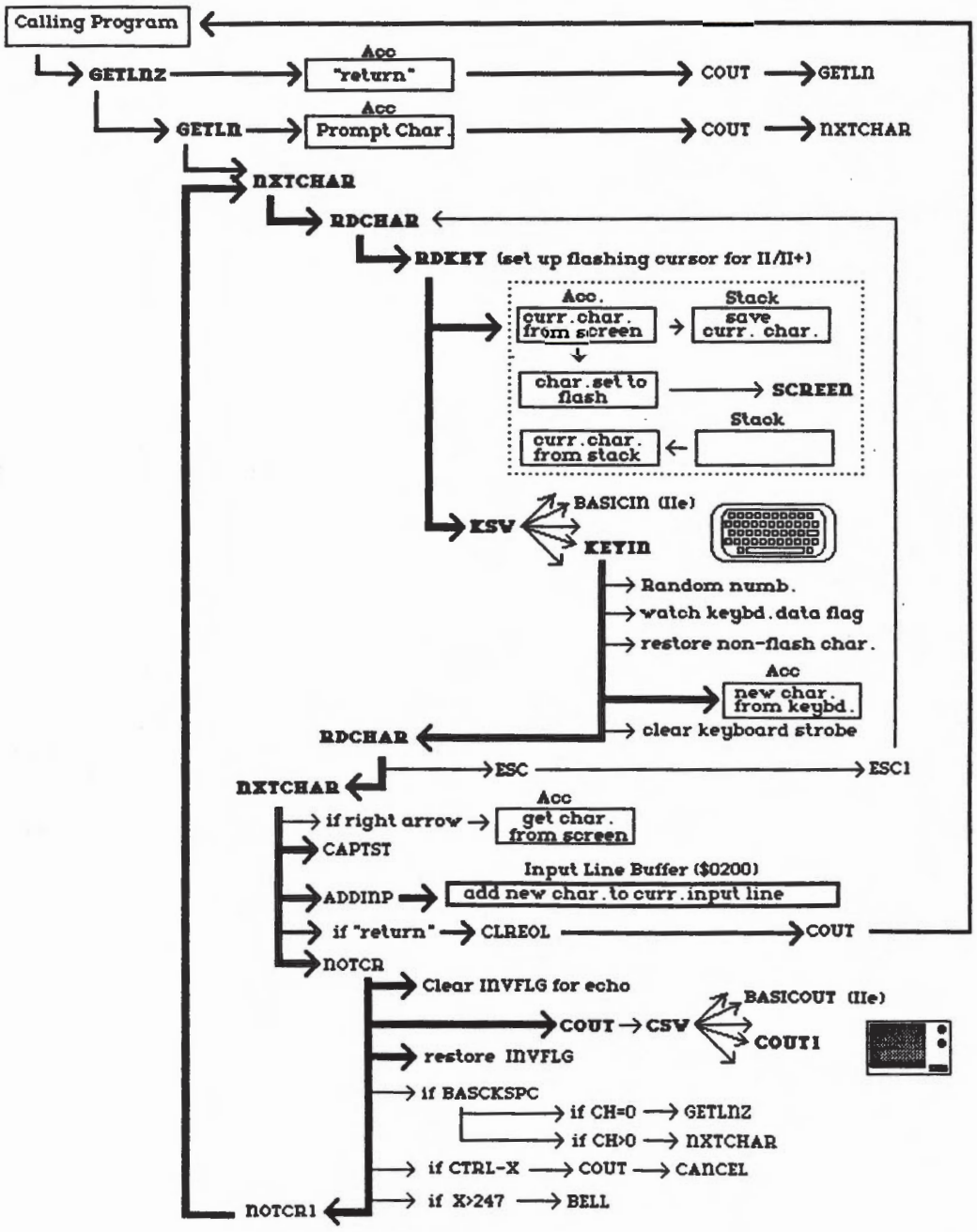


Fig. 33.1 GETLN flowchart.

Going Through GETLN Step-by-Step

The first entry to this whole apparatus is often made at GETLNZ ($FD67 for those who are following along with their commented Monitor ROM listings in hand). All this does is to kick things off by sending a carriage return to COUT so that you get started at the beginning of a fresh new line. You then drop immediately into GETLN ($FD6A).

GETLN itself really does just two things. First, it finds out which prompt it should be displaying (square bracket for Applesoft BASIC input, asterisk for the Monitor, question mark for input statements inside a BASIC program, etc.) and it tells COUT to put that prompt on the screen. Assuming we came in through GETLNZ, COUT has just finished doing a carriage return so the prompt ends up in the correct position at the beginning of the new line. The second and final task for GETLN is to send the 6502 into the NXTCHAR loop from which it will not emerge until it has captured a completed line of input.

NXTCHAR ($FD75) has a whole bunch of important subroutines, but its tasks can be roughly divided into first actually grabbing the next incoming character, then seeing that it gets put where it belongs, and finally deciding whether to go back for more characters or to return to the calling program to turn over the new data. The character grab is done from within RDCHAR ($FD35), but RDCHAR is not supposed to pass the character along if it's an Escape key. RDCHAR therefore has two parts; one which passes on the call for a new keypress, and one which checks for the escape code.

To actually start the character grab sequence, RDCHAR calls RDKEY ($FD0C), and RDKEY announces to the human that it is ready for a little action. To get the human's attention, RDKEY looks to the next position in text screen display memory and copies in that character. The character is stowed away for safekeeping, and then RDKEY calculates the proper value to get a flashing version of the same character and stores that flashing version in screen memory where it originally found the character. This is how RDKEY puts a flashing cursor on the screen at the position where you're supposed to type next. The checkerboard cursor in the //e 40 column mode is based on the delete character and the flashing is handled in a special way from software.

Apple //e owners may know that it can't work like this in the //e 80 column mode since there are no flashing characters available. RDKEY, however, doesn't know this and tries to do it anyhow—we'll see shortly how this gets straightened out. It is also worth mentioning that in the 80 column video boards for the II/II+ and in the Ultraterm, the flashing cursor is generated by special hardware on the video board (see Chapter 6), so the II/II+ RDKEY system is a bit unique.

The KSW Input Switch

With the cursor flashing, RDKEY has done all it can to get your attention, and so it then takes steps to plunge the Apple into the potentially endless loop of KEYIN. However, there is an intervening step here which is very fundamental to understanding the Apple. Rather than calling KEYIN directly, RDKEY does an "indirect jump through KSW." The idea of an indirect jump is covered in Chapter 27, but to review, the 6502 looks at the numbers stored in a location called KSW, and uses these to find out where to send control.

Key Switch (KSW) is the name for a very special pair of zero page locations, $0038 (KSWL) and $0039 (KSWH) (decimal 56 and 57). You see, the RDKEY routine doesn't really care how the next character gets into the Apple. It dives blindly through KSW and assumes that when control reemerges there will be a new character deposited in the 6502's accumulator ready for RDCHAR to take a first look at. However, the flow of control that plunges in through KSW, may be directed to anyone of dozens of possible subroutines.

In the standard Apple II, when turned on, KSW is set to point to the KEYIN routine. However, if you're using an external terminal, or a modem, or even a special video board but with the standard Apple keyboard, the contents of KSW will be changed to give control to some different and specialized subroutine. The most convenient example of such an alternative is the BASICIN entry point for the //e 80 column system. If characters are supposed to come in the age old Apple way, the contents of KSW are $FD1B, the start of KEYIN. If the //e 80 column card is active, however, the contents of KSW is $C305, the start of a quite different path. Whatever routine is actually selected via the jump through KSW, all end with an RTS (return to subroutine) instruction which returns control to RDCHAR for the Escape check.

There is a similar location called CSW which guides the execution of the COUT routine. This normally points to COUT1 (described in the next section), but can also be altered to point to any other routine.

Looping Inside KEYIN and Operating Some Hardware

The KEYIN routine in the II/II+ is very simple. In fact, there is so little to do that Wozniak threw in a little extra work for KEYIN in order to get something worthwhile out of all the treadmilling going on in there. KEYIN does double duty as the Apple's pseudo-random number generator. As it loops round and round waiting for a keypress, it increments a number stored in the zero page called RND ($004E, $004F) each time it loops. The assumption is that the actual microsecond at which any given keypress arrives will be fairly random, so the number left here when you're out of KEYIN should be unpredictable.

Other than that extraneous (and problematic) nonsense, what KEYIN actually does is to watch the contents of the eighth bit in the byte currently available at the keyboard input port (see Chapters 8 and 26). The keyboard port really has two parts. The lower seven bits store the ASCII code of the most recently pressed key, but the eighth bit is used as a keyboard data flag. When you press a key, the flag bit is set to one, and this signals KEYIN that it can grab the character and head back to RDCHAR.

However, before KEYIN leaves the area, it sets the flag bit back to zero. When NXTCHAR sends KEYIN back in to look for the next character less than a thousandth of a second later, it is not likely that you will have pressed another key yet. The value of your previous keypress will still be sitting in the port buffer. However, KEYIN will not be fooled by this since it had previously cleared the keyboard data flag. There will therefore be a zero in the eighth bit and KEYIN will ignore the ASCII code still sitting in the port, waiting instead for the flag to change before it grabs another character.

In the Apple //e, KEYIN has been expanded substantially. Shortly after arrival, the 6502 is told to hit the SLOTCXROM softswitch (see Chapter 26) and jump to the expanded version via an entry at $C100 (B.FUNC) which leads it to B.KEYIN ($C288). The extra routines here

make it easier for an external device to interrupt the KEYIN loop (important for ProDOS). This expanded version of KEYIN is only used when you are working in 40 columns and have not activated the 80 column firmware.

The //c has a single all purpose KEYIN entry, and although it performs the same fundamental functions as the II/II+ routine, it does several loops down into the I/O ROM beginning at $CC4C in order to accommodate the various text video modes.

Getting Input in //e 80 Column Mode

When an Apple //e or //c is first turned on, the machine launches into a reset routine (see Chapter 27) which, among other things, sets KSW to point to the standard KEYIN routine. However, when you type: PR#3 and hit Return, BASIC (or DOS) recognizes this as a command to activate the 80 column control program ROM. A number of things get done in this initialization process, but among them is the stuffing of a new value into KSW. This sequence also occurs when you type ESC 8 after a reset on a //c.

Thencefore, unless you type in an IN# command for one of the seven accessory slots or do the Escape Control-Q sequence to disable the 80 column board, KSW will point into 80 column control ROM. Whenever RDKEY tries to collect a keypress, its request will be routed through the 80 column ROM instead of directly through KEYIN.

PR#3 and Routes into the //e 80 Column Card Firmware

When the 80 column card is active, the Monitor's character I/O routines are still called in their normal way. It is only deep within GETLN, when it's actually time to jump through KSW or to do a COUT, that control shifts temporarily to the 80 column control ROM. The 80 column firmware is stored in ROM on the //e motherboard, and is in usually operated just like the ROMs on peripheral cards in the Apple's accessory slots. This control procedure is reviewed in Chapter 22 but, for the 80 column card, it guarantees that the 256 locations between $C300 and $C3FF will be available to the 80 column card and that it can grab temporary hold of the 2K of addresses between $C800 and $CFFF.

There are three entrances to the video control program in the //e $C300 ROM, placed one after another quite near each other. One, at $C300 (BASICINT), leads to an initialization routine. A second, at $C305 (BASICIN), is the entry point for accepting keyboard input, and a third, at $C307 (BASICOUT), is the entry point for routines which want to send characters to the screen. The prefix BASIC on all of these //e entries reflect the fact that other routines in the ROM are used by the Pascal operating system I/O routines. The analogous entries in a //c are C3ENTRY at $C300, C3KEYIN at $C305, and C3COUT1 at $C307.

What BASIC (or DOS) actually does when it sees a PR#3, is to set CSW to contain $C300, and then it simply goes back to GETLNZ. When GETLNZ tries to send its carriage return to COUT, it ends up jumping into the $C300 initialization routine (BASICINT on the //e, C3ENTRY on the //c). Only after initialization is complete will the carriage return actually be passed along to BOUT, the //e 80 column card equivalent of COUT1. Before the sequence of events passes from GETLNZ to GETLN, the full initialization process will be complete. Your PR#3 command will have been carried out and your next keypress will be displayed in 80 column format.

Standard Warmup for the //e 80 Column Firmware

On entry via BASICINT, BASICIN or BASICOUT, the 6502 sets one of three flags, one assigned to each of the three entries. These flags permit the 6502 to check back later to find out what it's doing out there in the $C300 space. No matter which of the three entrances is used; however, once the appropriate flag is set, the 6502 always branches to a general entry program called BASICENT ($C317). This routine siezes the 2K of address space between $C800 and $CFFF (see Chapter 22) and shoves 2K of additional 80 column firmware routines into it. Also, BASICENT stuffs a few critical parameters into the "screen holes" reserved for slot 3 (see Chapter 21, Figure 21.6b).

Those of you who read through the memory chapter may recall a few obscure locations in the Apple called screen holes. These holes are a result of the fact that the Apple reserves a full 1,024 ($400) locations in memory for storing characters for the 40 column text display, but that there are only 960 characters to store (40 by 24). The 1024 minus 960 equals 64 unused memory locations are available as little chunks of scratchpad RAM for use by peripheral cards. They are called screen holes only because they're mixed among screen display memory locations. It is incidental that they are used by the 80 column cards as scratch pad RAM for managing the screen display.

BASICENT has names for some of the screen holes such as CHAR (into which it stuffs the character that it is working on), MODE (which it is constantly updating to reflect what task it is currently engaged in), and OURCV, OURCH, and OLDCH (in which it keeps information about the vertical and horizontal cursor position). Once the screen holes have been stuffed, and space has been expropriated for the additional 2K of ROM, BASICENT checks back to see who called it—an initialization entry or an I/O entry. If it's an initialization entry, it jumps to BASICINIT in the $C800 space (specifically, to $C803). BASICINIT is in the $C300 space at $C317 in the //c.

Full Initialization With PR#3

The BASICINIT initialization routine includes checking to be sure it's got a good //e Monitor ROM out there (and it jumps ship by hanging the computer if it can't find the right ROM), and stuffing a few more screen holes with key parameters. But more to the point, it stuffs $C305 into KSW and $C307 into CSW. This gets done between the very first GETLNZ and GETLN after you have entered your PR#3 (see above).

The final task in the initialization process is to set up several of the video softswitches (see Chapter 26). It checks to be sure there is a card in the auxiliary slot, since it is possible to call and use these routines even if there is no card there. If it finds the card to be present, it selects the alternate character set (no flashing characters), and informs the video display generator that it will have to start scanning a full 80 column display. Finally, it clears the screen display memory to get rid of any extraneous junk that might have gotten into RAM before startup, sets up the scrolling operation, and outputs the carriage return which GETLNZ was trying to send when it got waylaid in the first place by the $C300 your PR#3 put into CSW.

BASICIN Instead of KEYIN in the //e

Now that everything is initialized, and KSW is set to point to BASICIN ($C305), it's OK to go back to where you were reading about RDKEY and its jump through KSW to the keyboard input routine. On arrival at BASICIN, the 6502 has to go through a variety of tasks before it actually gets to GETKEY (the true //e equivalent of KEYIN). First, as described above, it must go through the warmstart procedure in BASICENT.

On finishing up in BASICENT, in does not go to BASICINIT, but rather heads for C8BASIC ($C866). This is a secondary warmstart routine which is used by calls through both CSW and KSW. Here, the 6502 checks to see if it will be doing 40 column or 80 column work, and sets 80STORE (see Chapter 26) accordingly. There is some checking about the presence of the proper Monitor ROM and to make sure things are all set to handle cursor position correctly. Finally, the //e's 6502 checks the entry flags and decides whether to head for BOUT ($C896) to send a character to the video system, or to go to BINPUT ($C8F6) to check for keypresses. In the //c, an entry to C3KEYIN at $C303 does nothing more or less than causing a JMP back to KEYIN.

The //e 80 Column Cursor Does Not Flash

You will recall that just before diving into KSW, RDKEY had just finished trying to generate a flashing cursor. But you will also recall that in the //e 80 column initialization process, the alternate character set got turned on, and thus there are no longer any flashing characters available. Worse, the system for setting up flashing characters involves fiddling with the two highest bits of the Apple ASCII code, and this can cause some slightly bizarre effects in the alternate character set.

Therefore, one of the first things BINPUT does is to undo RDKEY's attempt at generating a cursor by putting the stored original character back in it place. It then uses its own cursor position information to generate a simple inverse cursor. This is actually a rather unfortunate solution to the problem because some people can't stand to work without a flashing cursor. These folks are balanced out by others who fear they'll go into an epileptic seizure if they are forced to watch all the constant flashing; but unlike the 80 column boards for the accessory slot, these people aren't given any choice in the matter.

The only possible solutions require major surgery to alter the firmware GETKEY routine to simulate a flashing cursor. If you absolutely must have a flashing cursor in 80 columns, you're going to have to buy an Ultraterm board (see Chapter 6), since it's the only one that lets you keep your extra memory, etc., in the auxiliary slot.

GETKEY and //e Escape Functions

The actual GETKEY routine ($CB15) is called from B.INPUT ($C905), and is a fairly close copy of the II/II+ KEYIN, doing the RND fiddling and a different version of the flag checking. However, instead of heading out of the $C800 ROM to RDCHAR in the Monitor, it comes back up into B.INPUT.

B.INPUT then handles a task normally left for RDCHAR. It does its own scan for escape characters. There are a large number of screen escape features that are new in the //e and the routines for all of these features take up quite a bit of room in the $C800 ROM (see Chapter 18, Table 18.2).

When B.INPUT detects an Escape keypress, it calls a routine called ESCAPING which routine calls GETKEY again to get the second key in the sequence (for instance, the M in the sequence Escape M). When the second keypress arrives, ESCAPING uses it to choose an escape screen subroutine. All succeeding keypresses are siezed by ESCAPING until it notices a character which doesn't call any of its routines. Only then does it relinquish control of the GETKEY routine and send the 6502 back to B.INPUT. If the incoming character is not an escape key, B. INPUT sends control to NOESC ($C9B7) instead.

The Right Arrow for //e Screen Picking

The second task undertaken by B.INPUT is to handle the screen pick (right arrow) function. This is actually a bit subtle and involves more than just moving the cursor forward one space. As most folks know, you can use the escape functions to move the cursor to any point on the screen, then use the right arrow to copy characters from the screen display into the line buffer. This is very handy, for instance, when you're looking at a long filename in a catalog—just type RUN, then escape your way up to the file, and start hitting the right arrow to enter the file name. It is also used extensively in editing Applesoft programs.

The screen pick function assumes that the cursor is already positioned where you want to get your character, and then it actually siezes GETKEY's place in the normal hierarchy. Each time you hit right arrow key, PICK goes out to display memory and grabs a character. This character gets fed into the character stream as if it had come from the keyboard.

No Screen Picking from BASIC

This is fairly straightforward; however, BASIC programmers can go crazy trying to figure out exactly what is going on with this right arrow in the //e, all due to some nasty twists and turns thrown in. To sort it out, you need to pay attention to two ASCII values; one is $15 (CTRL-U) and the other is $1C (FS). When you press the right arrow key, it always sends a CTRL-U to the GETKEY routine, but what happens to that CTRL-U varies.

If GETKEY was called from ESCAPING, then the CTRL-U gets abruptly changed into a Forward Space (FS). This trap makes the right arrow look like an Escape "K"—which also gets translated into an FS. When the escape mode is active, K is equivalent to the right arrow, and both send an ASCII $1C for handling (you can also type this code in as a CTRL-\). You get a forward space, but no screen pick.

If, however, GETKEY was called from B.INPUT, then the $15 (ASCII CTRL-U) gets passed along to NOESC where the pick function is carried out. The final confusing wrinkle is that if you type CTRL-U from the keyboard, you get a screen pick (since you were actually using B.INPUT). However, if you try to send a CTRL-U [CHR$ (21)] from a PRINT statement in BASIC, your 80 column display rudely and abruptly shuts down and you get bounced out into 40 columns.

Input Characters Versus Output Characters

This very different response to CTRL-U when PRINTed from BASIC points up a crucial distinction between character input and output. During keyboard input, all characters get examined by B.INPUT or RDCHAR, or some such part of the input chain. During input, the COUT routines get called to let you see what you are typing, but they are really a whole separate realm to themselves. When a BASIC program executes a PRINT statement, it sends characters directly to COUT, without using any of the input routines. Therefore, when you try to PRINT a CTRL-U, it never gets checked by ESCAPING or by NOESC. It therefore has no way of activating either the screen pick or the FS routine.

Further, no CTRL-U that gets presented to the input routines ever makes it out of the input loops. It either gets used for cursor positioning or replaced by the character picked from the screen, but in no case does it get put into the line buffer or sent from the input routines to COUT. Therefore, the Apple makes a clean distinction between CTRL-Us that show up as input from the input routines and CTRL-Us that are output from a program (i.e., PRINT CHR$(21. Input CTRL-Us have been assigned to forward space and screen pick, while output CTRL-Us have been assigned to a BOUT routine called QUIT which shuts down the 80 column card. (You use Escape CTRL-Q to get to QUIT from BINPUT.)

Literals and Uppercase Restrict

The last thing that happens in the BINPUT chain of events is a series of tests to find out whether characters should get changed to uppercase. This has to do with a frustrating problem that arises while you're entering Applesoft program lines in the II/II+ and //e. The Applesoft Interpreter insists that all characters be in uppercase, yet within the quotes of PRINT statements, it's OK to mix uppercase and lowercase letters.

In the //e, Apple has included a "restricted-case mode" which can be invoked by typing ESC R. While this mode is active, the input routines turn all letters into uppercase unless they're between quotes. This is all done as part of the BINPUT routines and so the characters that get passed back out to RDCHAR have been forever modified before any other program can get a look at them.

Getting Back to RDCHAR

In the Apple II/II+, the few bytes of the KEYIN routine are followed immediately by a return to RDCHAR. The whole mass of BASICIN, BINPUT, GETKEY, etc. stuff in the //e heads back to a standard exit routine called BIORET ($C8E2) (the name means that both BASIC input and output leave the $C800 ROM via the same return routine).

In all cases, the 6502's accumulator now holds a new character. This character may have come from keypresses in KEYIN, B.KEYIN, or GETKEY. It may have come from a //e 80 column card screen pick, and, for that matter, it may have come from any input device whatsoever that was pointed at by KSW. All //e escape routines have already been executed. RDCHAR's job now is to get the character headed on its way for full analysis.

RDCHAR itself has just two options here depending on whether or not the character is an Escape. Although B.INPUT handles the escape calls from GETKEY, any that came in via KEYIN or B.KEYIN must be handled here. RDCHAR responds to Escape sequences much as described for B.INPUT. If there's an Escape, it calls RDKEY again to see which escape function to execute. In the //e, these second inputs get fiddled around a little via some $C100 routines, but these just serve to condition them properly for RDCHAR (for instance turning the up arrow into the proper form of Escape I, handling lowercase i, j, k, m, etc.).

Once RDCHAR has carried out one of its small set of escape functions, it calls RDKEY once again to get things back on track. Any incoming characters that are not part of an escape sequence get passed back on up to NXTCHAR. NXTCHAR handles any CTRL-U screen picks from KEYIN or B.KEYIN, and then passes through a famous II/II+ subroutine which converts lowercase ASCII codes to their uppercase equivalents. This CAPTST step is removed in the //e and //c.

Processing by NXTCHAR and NOTCR

All characters now arrive at a step within NXTCHAR called ADDINP ($FD84) which is the pivotal step from any other program's point of view. Here, the character is put into its correct place among the 256 positions in the Input Line Buffer in RAM memory between locations $0200 and $02FF (decimal 512 to 767). As you type within the GETLN system, this is your principal objective; getting characters into the Input Line Buffer where a program can analyze your line of input once you've hit return.

After safely depositing your character in the Line Buffer, NXTCHAR attends to echoing your input to the screen so you can see what you're doing. Here, its key question is whether or not you just hit return. If you did ($0D in the accumulator), then a "Clear to End of Line" command is issued to the screen routines, and the carriage return is sent to COUT. In the //e, all that's required is sending the return to COUT, and the firmware does the clear automatically.

When COUT finishes the necessary screen management for the carriage return (including scrolling if necessary), program control goes back to the program (such as the Applesoft Interpreter) which called GETLNZ in the first place. That program can now grab the contents of the Input Line Buffer and decide what actions to take in response to what you typed in.

If, however, you're not yet finished typing in your line, then something is going to have to be done about displaying the character. This is all handled by the various parts of NOTCR ($FD3D). In the II/II+, NOTCR sets up the INVFLG mask to force all of its echoed outputs into the normal rather than flashing or inverse character sets. Then, with the character in the accumulator, it does a JSR to COUT ($FDED) which causes the 6502 to jump through CSW to the appropriate output routine. In standard 40 column mode, this routine will be COUT1 ($FDF0); for //e 80 column work, it will be the BASICOUT entry to the 80 column firmware at $C307. And, of course, it may be directed instead to a printer, or to a printer and then the screen, etc., via various PR# statements and internal calls.

Standard Output with COUT1

All characters arriving at COUT1 are supposed to have a 1 in their eighth bit. This is called excess $80 format, and must be attended to by other devices which are used for input and output in the Apple. When characters come from the keyboard, their eighth bit is set because of the keyboard data flag (see Chapter 8) in the eighth bit. COUT1 first checks to see if the character for output is a control character, and if it is not, it uses the contents of INVFLG to alter the character for inverse or flashing display (see Chapter 26, Table 26.6).

In all but the earliest Apples, COUT1 then passes through a routine called VIDWAIT ($FB78) which lets someone at the keyboard halt the display process. Basically, it just takes a quick read of the keyboard, and if there is CTRL-S in the keyboard input buffer everything stops, and the VIDWAIT goes into a holding loop waiting for something else to be typed. You can try this by loading a BASIC program and giving a LIST command; while BASIC is using COUT1 to print the program to the screen, a CTRL-S will halt the listing. Notice that there is no cursor, because you are nowhere near the RDKEY or BINPUT routines which generate it.

If you type any character now, VIDWAIT will see the keyboard data flag change and burst out of its waiting loop to continue with COUT1. Usually, VIDWAIT also clears the keyboard data flag so your keypress to restart things doesn't look like real input the next time NXTCHAR looks at the keyboard. If you type CTRL-C, however, the flag is not cleared, and your keypress will be read to break out of the listing routine completely.

The character for output is then checked for special handling. If it's a carriage return or a line feed, the scroll routines get called up, and if it's a backspace, a simple change in the

horizontal cursor position is carried out. Otherwise, all characters get sent to STORADV ($FBF0). The 6502 then checks a location called BASL (at $0028, decimal 40) where the current vertical position in the screen display memory is kept and also checks CH (at $0024, decimal 36) where the current horizontal displacement is stored, and does an STA (store the accumulator) to move the character from the accumulator to its proper position in display memory. Shortly thereafter, the video display generator will see it there and it will be painted on the CRT screen for you to see.

There is a little bit of calculation done now to see if BAS needs to be changed (move to the next line for the next character) and to figure out a new value for CH. These results will also dictate whether or not the whole screen should be scrolled. With all this complete, there is an RTS which sends control back out to the caller, in our case, to NOTCR in the GETLN system (see Figure 33.1).

BASICOUT in //e 80 Column Mode

With the //e 80 column card initialized, CSW points to BASICOUT ($C307) in the 80 column firmware. The 6502 goes through the common entry steps described above including BASICENT and C8BASIC, but finally it heads out to BOUT at $C896. There, a few checks are done very much as in COUT1, and a routine similar to VIDWAIT is passed through. After this, however, BOUT has to do a more elaborate scan for control characters in the output stream because it has a greatly expanded array of control character sequences relative to the II/II+ (see Chapter 18, Table 18.2). If it's not a control character, the 6502 heads for STORCHAR at $CEF2.

STORCHAR takes care of any requests for inverse or normal character output and then leads into the SCREENIT routine ($CF06), which has to handle bank switching of the display RAM to get the 80 column stuff right. Those who have worked through the video chapter will recall that the 80 column display is constructed from two RAM display storage areas; one on the motherboard, and the second in 1K of RAM on the auxiliary card. The thing is that the two banks share the same address space.

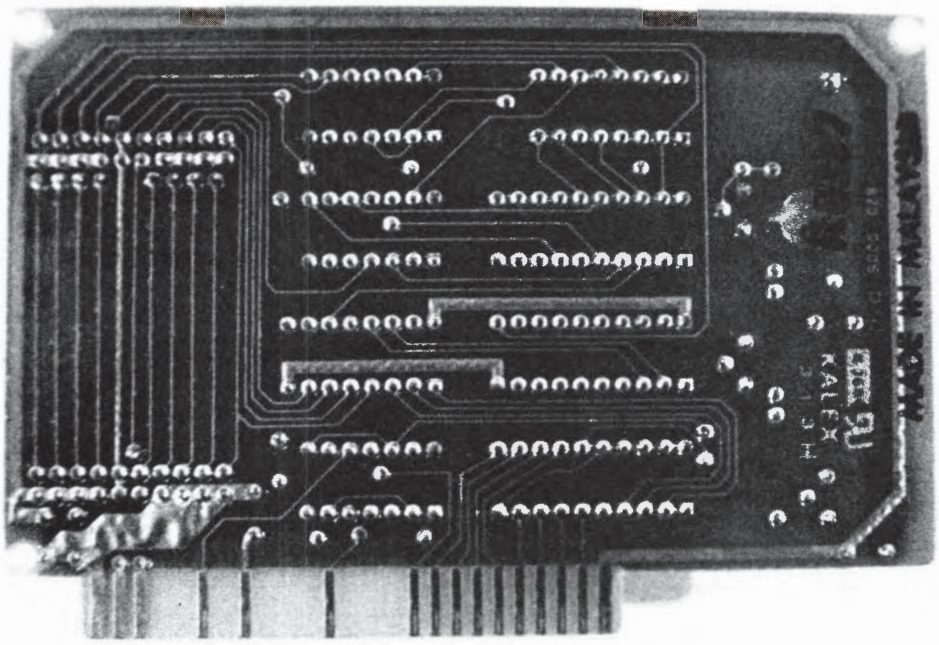
Of the 80 columns, characters in even columns are stored in the display memory on the auxiliary card while those in odd columns are stored on the motherboard. Now recall that in 40 column mode, the value of CH is used to calculate the actual memory position into which the character should be deposited. However, because of the bank switching arrangement, each of the 40 even/odd pairs have a shared address.

This is all resolved in a crucial step at $CF10. Up to this point, a cursor position between 0 and 79 has been maintained in CH. Here, a logical shift right (LSR) is performed which has the dual result of dividing CH by two so that it will point to one of the 40 even/odd pairs, and also setting the microprocessor carry flag to indicate whether this was an even or an odd CH at the outset. The 80-COL PICK routine at $CF14 uses the carry flag to decide how to toggle the PAGE2 softswitch ($C054/$C055—see Chapter 26), and the result of the division to set the horizontal address. Control then heads back to BOUT where scrolling is looked into, and then, finally, there is an exit through BIORET to NOTCR in the GETLN routine.

Managing the Input Loop

With control back at NOTCR, the chief remaining task is to decide if it's OK to loop back up to the top of NXTCHAR to collect another keypress. First, NOTCR checks to see if a backspace (CTRL-H) was just output. It it was, and if this pushes the cursor to the left margin of the screen, then NOTCR sends control all the way back to the entry at GETLNZ in order to get a carriage return and a new prompt character.

It also checks for a CTRL-X which tells it to cancel the line and junk the stuff in the line buffer. Finally, it keeps track of how many passes through the loop have been made. After the 247th pass, it causes a beep after each keypress, and if you don't heed the warning, it issues a cancel and wipes out your line. There are only 256 places in the Input Line Buffer and NOTCR will not permit you to overrun this limit. If none of these contingencies arise, then you will be sent back up to the top of NXTCHAR for a new cursor and a new character of keyboard input (see Figure 33.1).



Chapter 34

Sector I/O and File Management with DOS 3.3

Disk Sectors and File Managers

When a program is loaded and running inside the Apple, it is usually organized as a single long, continuous and coherent bit pattern in the motherboard RAM, stretching uninterrupted across tens of thousands of bytes of memory. However, when that same program is stored on a disk, it is fragmented into hundreds of scattered and lifeless shards.

But, scattered and lifeless though they are on the disk, all is not lost, for File Managers exist to put them all back together again the next time they are needed inside the computer.

The reason for all the scatter is that the storage of data on disks has a lot more to do with say, finding a parking place in Greenwich Village on a Saturday night in July, than it does with neatly filing books away on the shelves of a quiet library. It would have been nice if you and your friends could park in two consecutive parking spaces directly in front of your favorite restaurant, but you are dealing with competition for space on the part of other folks who hold nothing against you but whose first priority is to get their car off the street as quickly as possible, preferably in front of their restaurant.

You're fine if you don't mind walking a little distance, but of course you'd better be certain to remember exactly which side street you parked on. Disk "directories" help file managers remember where they left their sectors. These directories are not of trivial importance. As all too many of us have learned, a disk with 120K of good data but just a little touch of clobbered directory is one lifeless disk suitable mostly for depositing in a trash can.

The Shape of a File on Disk

As long as a program is neatly assembled in memory as a single continuous range of bytes, the File Manager can describe the program to its complete satisfaction with a name to identify it, its starting location in memory to know where to look for it, and its total length in bytes. When the same program is stored on disk, the File Manager needs the name, the location in memory where it should put the program when it is reloaded, it's length, and also a description of its physical layout on the surface of the disk.

The physical organization of data on disks is described in detail in Chapter 23, so a brief review will have to be enough here. The fundamental storage unit on an Apple disk is a "track" which can store about 4K of data. Before each read or write operation, a single "read/write head" must be physically moved into a track.

There are 35 tracks on an Apple diskette, and it takes 20 milliseconds for the head to travel from one track to its neighbor, and hence 700 milliseconds to travel from the innermost track to the outermost. On arrival at a selected track, the read/write head can begin to read the data stream as the bits on the track spin by. The data track is divided into 16 segments, each containing 256 bytes, called "sectors." The disk spins at 300 revolutions per minute, so it can take up to a maximum of about 200 milliseconds for a given sector to make a complete spin and come back around to the read/write head.

Time and Space on the Disk Surface

In an ideal situation for the quickest possible read, you would move the head to one track, read in 16 sectors of information, step the head over into the next track, read in 16 more sectors, etc., until the whole file was read in. On the other hand, in the worst case, once you arrived at the track you would have to wait for the one single sector you needed to come by, then step over 10 or 20 tracks and wait for your second sector to spin by, etc. In the first case, you could spend about a total of half a second in mechanical head moves and waiting time for the spinning disk while you collected 32 sectors. In the second case, you could easily push that up to 20 full seconds to read in your 32 sectors (8K of data).

In real life, the sectors of a file are usually clustered together in clumps requiring a relatively small number of moves; however, as you write and rewrite several different files on the same disk, the sectors tend to get intermingled. In order to retrieve the intact file, therefore, the File Manager must have a list of the sectors used to store the contents of each file on the disk. That list of sectors must also tell the File Manager the order in which to read the sectors to get the file properly reassembled.

Each Apple disk sector can be uniquely identified by its track number (between 0 and 34), and its sector number (between 0 and 15). This track/sector (T/S) pair is that sector's address, and in fact, the appropriate T/S address is recorded in the beginning of each sector during formatting. Each sector therefore has an "address field" with its T/S identifier, and a data field for 256 bytes of information. There are additional complexities described in Chapter 23, and if you really want to know more, you should get *Beneath Apple DOS* by Worth and Lechner.

Apple DOS 3.3: Names in the File Descriptive Entry

In sum then, a File Descriptive Entry must tell the File Manager the file's name, where it should be loaded in the Apple's memory, how long it is, and the order in which it should read specific disk sectors to retrieve it. The first of these, the name, is fairly straightforward. This will be a string of ASCII characters which are meaningful both to the File Manager and to the human who is calling the shots. The only particulars will have to do with how long the name is and if the name can contain any non-alphabetic characters such as semicolons and control characters. In Apple DOS, you are allowed up to 30 characters with the condition that the first character be a letter, and that you use no commas.

The length given in a File Descriptive Entry is actually a count of how many sectors your file takes up. This value is displayed on the screen when you type CATALOG and helps you keep track of how fast you're using up space on the disk.

Apple DOS File Types

The information on the destination in memory is a little more subtle. All Applesoft programs get loaded into Apple memory at a standard beginning point determined by the Applesoft Interpreter, usually $0800 (decimal 2,048), although you can change this (see Chapter 40). The File Manager therefore simply marks the File Descriptive Entry to show that the named file is an Applesoft program. When it is asked to load the program, it gets the starting address from the Applesoft Interpreter.

Machine language programs have no standard starting location in memory, so the File Manager assumes that they should get loaded back into the same place where it first found them. To carry this out, the File Manager stores that starting location as part of the file, and marks the File Descriptive Entry to show that the file is a "binary" file. To load a binary file, the File Manager first checks to find the starting address to use for the destination, and then goes on with the load.

A third type of file is called a text file. These files have no obvious destination in memory, and, in fact, they are rarely, if ever, assembled into complete ranges inside the Apple. Rather, each small part of a text file has some importance to a program already running in the computer. The emphasis in text files is not on where to put them; instead, it is on helping a program to find the one small part it is interested in using at any given time.

In Apple DOS, text files are marked off into records whose length is quite independent of sector size. The File Manager is prepared to help a program find the record it is looking for and to find the particular bytes it needs within the record. However, since the internal structure is determined in part by the program which wrote the text file, the File Manager must work with that program in order to use the signals within the body of the data to find their way. The File Descriptive Entry is simply marked "T" for text. This is why DOS text files can only be read from within a program.

Most word processors and database programs have their own styles for keeping track of the internal structure of a text file. There are, however, a number of numerically oriented Apple DOS programs such as VisiCalc which use a standardized internal system called a DIF text file format (see *The DIF File* by Donald H. Beil (Reston. Both CP/M and ProDOS provide additional information along these lines in their file descriptive entries, and each of these operating systems is therefore able to load and display a text file even when no program is running.

The Track/Sector List

The crucial remaining element in the description of the file is an ordered list of the sectors in which the file is stored. In Apple DOS, this list is kept separate from the rest of the File Descriptive Entry and is actually stored in its own sector on the disk. The File Descriptive Entry contains only the address of that Track/Sector List sector.

This is a concept which may be familiar to those of you who plowed through the sections on machine and assembly language. The File Descriptive Entry contains only a pointer to its T/S List. The File Manager first reads the File Descriptive Entry to learn the address in that pointer, then it uses the pointer to find and read the T/S List. Only then can it actually begin to read the information in the file itself.

Track/Sector Lists as Directory Sectors

This introduces another crucial concept. Some sectors on a disk are used to store files. Other sectors are used to store descriptions of files. These file descriptive sectors are usually referred to collectively as a disk directory.

A T/S List Sector is a 256 byte sector physically identical to any other sector on the disk. In each of these T/S List Sectors, 12 bytes have been reserved for other record keeping tasks, and the remaining 244 bytes are taken up by the track/sector pairs which describe the file.

Within the T/S List Sector, each pair takes up two bytes; one byte for the track number and one for the sector number, so there is room for 122 of these pairs in a single T/S List Sector. Since each pair points to a single 256 byte data sector, the entire 122 pair list can describe $122 \times 256 = 31K$ bytes of program or text. If a stored file is longer than 31K, then a second T/S List Sector must be set up, a third if longer than 62K, etc.

The File Descriptive Entry for these large files still points only to the first T/S List Sector. It is the responsibility of that first T/S List Sector to point to the second T/S List Sector for that file, and so on. This is called a linked list arrangement (see Chapter 35, Figure 35.1). Just to be safe, each T/S List Sector in the linked list also has a notation stating whether it starts with the 123rd, 245th, etc., pair.

Catalog Sectors in DOS Directories

The File Descriptive Entries described so far are actually laid out as follows: two bytes point to the T/S list, one byte for the file type, 30 bytes for the name, and two bytes to describe the length of the file; for a total of 35 bytes. These File Descriptive Entries are stored on the disk in Catalog Sectors. A 256 byte Catalog Sector has room for seven of these entries ($7 \times 35 = 245$) with 11 bytes left over (see Chapter 35, Figure 35.2). The descriptive entries for the first seven files put onto a disk go into the first Catalog Sector. After that, a second Catalog Sector is set up for the next seven File Descriptive Entries, and then a third, and so on.

The series of Catalog Sectors on a disk are laid out in a linked list pattern. Each Catalog Sector has an address pointer which can tell the File Manager where to find the next Catalog Sector. As long as the File Manager starts by reading the first Catalog Sector, it can find its way to the second, and so on. On Apple DOS diskettes, these Catalog Sectors are usually laid out along the seventeenth track (track $11 hex). The reason for the otherwise strange number, 17, is that this is exactly halfway from the inside to the outside of the disk. Every file access begins with a scan of the Catalog Sectors, and this arrangement guarantees that the read/write head will always be within 16 tracks of its destination when it finishes reading the Catalog Sector.

A Bit Map to Describe the Disk

If you glance at Chapter 35, Figure 35.1 and Chapter 36, Figure 36.2, you can get an idea of the sort of sprawl that a large directory can turn into. There is a linked list of Catalog Sectors, each of which contains up to seven File Descriptive Entries. Each of these entries point to a T/S List Sector, and, for larger files, these may also be linked lists.

This system is very effective for quickly getting ahold of the complete description of a single file. The File Manager starts to scan through the linked list of brief File Descriptive Entries, seeing seven at a time, and once it spots the file name it's looking for it branches off to get the T/S List.

However, this is extremely inefficient for some of the File Manager's other responsibilities. As a file grows while you write more program lines or enter text into your word processor, the File Manager must continually assign new sectors to your file. As it does so, it must make sure that it doesn't happen to assign you a sector which is already being used for something else. For instance, what if the paragraph you just finished typing got written in the sector that had been storing the T/S List for some other file. Goodbye other file.

To handle all this, the File Manager must have a complete map of the $35 \times 16 = 560$ sectors on the disk. It is possible for the File Manager to step through the entire directory, branch to each T/S list, and slowly build up such a map of used and unused sectors, but this is tedious. In fact, CP/M does just that, and the result is one of the most annoying weaknesses of that operating system. Fortunately, DOS 3.3 has a faster and more secure way of proceeding.

The File Manager maintains what is called a bit map of disk sectors. In this map, each sector on the disk is assigned a single bit position. The 560 bits in the map take up just 70 bytes, so they fit quite easily into a single sector. Whenever the File Manager needs to allocate a new sector, it reads in the disk's bit map, and checks to see what's available. It locates the most appropriate free sector, marks the selection in the bit map, and then rewrites the altered map back out onto the disk. The new sector is also added to that file's T/S list, but marking the bit map makes the allocation public and easily accessible.

This has the disadvantage of requiring many reads and writes of the bit map as sector allocation proceeds, but it has the advantage of greatly reducing the risk of accidentally overwriting a sector. Some DOS software such as the Screen Writer II word processing program modify DOS by cutting out this update step in order to speed up disk accesses. However, DOS users are accustomed to popping disks in and out of drives with relative abandon and if you do so at the wrong time while using Screen Writer, it will happily overwrite, destroy and mangle disk data. CP/M doesn't include an on-disk bit map, but it does check first before writing to make sure it knows what's going on (this is the cause of the annoying R/O Error statements in CP/M when you're trying to use PIP in a casual manner—see Chapter 35).

Volume Table of Contents

This bit map might be called a disk descriptive entry. In fact, it is kept together with several other pieces of information about the disk itself, all in a single sector called the Volume Table of Contents (VTOC). Along with the bit map, the VTOC has two bytes which keep track of the last track in which a new sector was allocated, and the direction that the File Manager should look in to find its next track for allocation.

Recall that the Catalog Sectors are on track 17. The first T/S List Sectors and data sectors usually get put on track 16, and when that fills up, the allocation process moves to track 15, etc., on inwards towards the innermost tracks. When they are filled, the File Manager starts in again at track 18 and works outwards.

There is space in the VTOC for a "volume number," a sort of name for the diskette. Few people seem to actually use the volume number on Apple diskettes although the volume number can be quite important in modified versions of DOS which operate hard disks. There are also a few bytes which tell how many tracks, sectors, and bytes per sector are on the disk. This information is the same (35, 16 and 256) for all standard Apple diskettes, and DOS doesn't do too well with any other kind of disk; but when the VTOC was first laid out, this data reflected high hopes for future modification.

The VTOC is always in sector 0 of track 17, and it is always the first sector that the File Manager looks at when it wants to scan the directory. To kick off the whole scan process, the VTOC also has a two byte pointer to the first Catalog Sector. The VTOC is therefore the keystone of the Apple DOS directory system.

When the File Manager is given the name of a program to load, its first action is to send the read/write head to track 17, wait for sector 0 to spin by, read in the VTOC, and get the T/S address of the first Catalog Sector. With that information in hand, the File Manager can begin to scan through the linked list of Catalog Sectors until it finds the file name it is looking for. The File Descriptive Entry clumped together with the file name points to the first Track/Sector List Sector for the file, and once that is loaded the File Manager can begin to load the file.

Sector Buffers in RAM

Although the File Manager devotes a large part of its attention towards managing the sectors on the disk and keeping the directory under control, it has also got to take care of a few things inside the Apple in RAM. During every transfer, whether it is a read or a write, the File Manager must set up a 256 byte page of RAM memory called a sector buffer.

During any transfer, an effective link is formed between the 256 byte buffer in RAM and a 256 byte sector on the disk. Once the transfer is activated, it does not stop until the complete contents of the buffer have been copied to the sector (a "write"), or the complete contents of the disk sector have been copied to the buffer (a "read"). To change one single byte of data on the disk, the appropriate sector is identified and read into a buffer in its entirety. The one byte is changed, and the whole buffer is rewritten back to where it came from.

Whenever the VTOC is read, it is sent to the VTOC Sector Buffer in RAM at $B3BB to $B4BA (46,011 to 46,266), and Catalog Sectors always get sent to the Directory Sector Buffer in RAM at $B4BB to $B5BA (46,267 to 46,522), each one overwriting the one that preceeded it in the buffer. When a file is read, however, the buffer situation gets a little more tricky.

Loading Files

The question of sector buffers for files brings us back to the File Manager's key role of reassembling fragmented files. To load an Applesoft or machine language program, the best and most efficient loading system would be to set up the first sector buffer at the location in RAM

where the program was supposed to start, for instance at $0800 (decimal 2,048) for an Applesoft program. Once the first 256 bytes of the program was loaded into RAM there, the sector buffer would be redefined one page higher in memory, i.e., at $900 (decimal 2,304) to receive the second 256 bytes and so on until the entire program was loaded directly into its ultimate destination location. CP/M loads its program (.COM) files in this fashon, and ProDOS loads in this fashion, but unfortunately, DOS does not do this.

In fact, when DOS itself is loaded into memory during the initial bootstrap operation, this sort of "incrementing sector buffer" is used. However, at all other times DOS uses a File Buffer system that is both more complex and much slower. There is a resulting boom business in DOS enhancement software which lets DOS use an incrementing sector buffer for program loads (i.e., Diversi-DOS, David DOS, etc.—see also *How to Break the Speed Limit* by Art Schumer in CALL A.P.P.L.E. in Depth: All About DOS), but for the most part, you must use the File Buffer system.

DOS Data Sector Buffers

The DOS File Buffer system is based on the requirements of text files rather than program files. When some applications program is running in RAM, the File Manager has no particular way of knowing where that program and its variables, arrays, etc., are located. When DOS itself is first loaded and ready to run, it marks off an area of RAM, from $9600 (38,400) on up to the end of the first 48K of RAM ($C000, or 49,152), for its own use. Everything from $0800 (2,048) to $9600 (38,400) is set aside as free space for programs (see Chapter 40, Figure 40.2). DOS agrees never to use any RAM in the free space, and the programs are constrained never to use any space in the DOS area.

Whenever the File Manager reads a sector from a file, it loads that data into a Data Sector Buffer within the protected DOS area. It then launches into a tedious byte by byte transfer of the data to its intended location. This is built around the use of a text file by a program in Applesoft BASIC. In BASIC, you get input from the keyboard character by character as it is typed by issuing an INPUT command. When you read from a text file on disk, you use the same INPUT command, except that the Applesoft Interpreter is directed to collect the data byte by byte from the Data Sector Buffer instead of from the keyboard.

This is all very nice for text files. It lets you give commands that seem to read and write individual bytes from a huge text file. When you want to load a nice long continuous program, however, the File Manager uses the same byte by byte approach to feed the whole program into its intended location. The Data Sector Buffer is filled with 256 bytes of the program; these are moved out one by one, then a new sector of data is read in to the same Data Sector Buffer to get doled out to its own intended location.

As a result of this little bottleneck, your hard disk may transfer data from the disk to the controller at five million bits per second, but once 256 bytes has arrived, the File Manager can only pass it along at about 4,000 bits per second. The fastest Applesoft Load is done by the Vista A800 disk controller (see Chapter 23), which not only dodges around the File Manager, but also disconnects the 6502 to use DMA (see Chapter 27). This method can load an Applesoft program at a rate of about 80,000 bits per second or 20 times faster than Apple DOS with a Disk II.

Opening a File

While data is moving back and forth between a file's buffer and one of its sectors, that file is said to be "open." When the File Manager opens a file, its task is to set up a 595 byte File Buffer, set aside 256 bytes within the File Buffer for the Data Sector Buffer, and then fill up the rest of the File Buffer with vital statistics about the file.

DOS usually has space set aside for three full 595 byte File Buffers between $9600 (38,400) and $9CF9 (40,185; see Figure 34.1). When it opens a file, the first thing it does is to allocate one of these buffers to that file by writing the file's name in the File Name part of the File Buffer. Then it launches into the series of events described earlier. It reads the VTOC, and then starts searching for the Catalog Sector which contains the file name. At the time the file name is located, the VTOC is stored in RAM in the VTOC Sector Buffer, the appropriate Catalog Sector is stored in the Directory Sector Buffer, and the File Manager has just gotten hold of the sector address of the T/S List Sector.

The T/S List Sector is the crucial object of the directory search. Once it is located, it provides a detailed description of which sectors the File Manager should read to get the file's data. The appropriate T/S List Sector is copied into a part of the File Buffer called, of course, the T/S List Sector Buffer.

At this point, the process of opening the file is complete. The File Manager has all the information it needs to access any data in the first 31K (see above) of the file on disk. Subsequently, the File Manager leaves the T/S List in its area of the File Buffer and the various data sectors get streamed through the Data Sector area of the File Buffer.

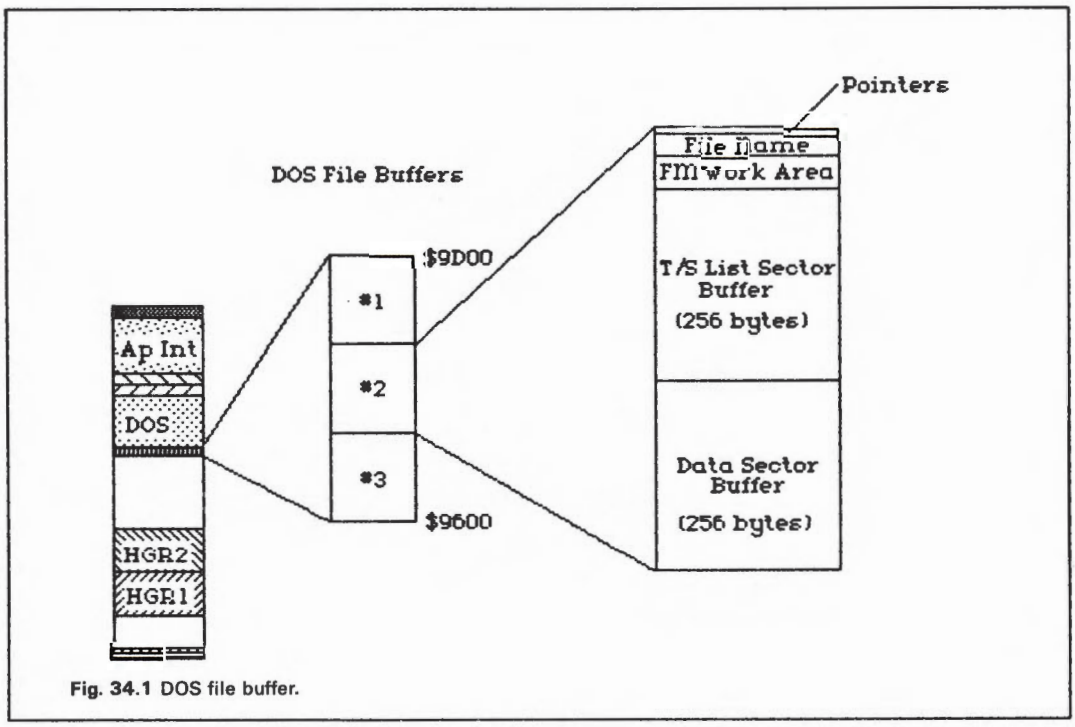


Fig. 34.1 DOS file buffer.

Calling RWTS to Operate the Drive

Fully armed with maps, lists, directories, and buffers, the File Manager has all the information and all the RAM it needs to do its work. To carry out the actual physical data transfer, the File Manager uses a Peripheral Management routine called Read/Write a Track/Sector (RWTS). The RWTS routine has various supplementary subroutines for error checking and data encoding, and also six principal hardware control subroutines which actually operate the disk drive interface.

When the File Manager is ready to put RWTS to work, it collects up all the pertinent information to fill in the variables in the RWTS Parameter List (at $B7E8, 47,080). This list is the standard version of the RWTS I/O Control Block (IOB). Once the IOB is filled in, the File Manager calls RWTS with a JSR $03E3. This is a jump through one of the page $03 vectors (see Chapter 21, Figure 21.5) and can be changed to use a different control routine. When RWTS is in use, the vector points to the RWTS entry at $BD00 (48,384).

The IOB must be filled in, and RWTS called, for each sector that is read from or written to. The parameters RWTS needs are: slot, drive and volume, track and sector, the address of a Device Characteristics Table which tells a little information about the drive, the address of the sector buffer to be used, a reminder about which drive it used last, and one of four commands. The commands are SEEK, READ, WRITE and FORMAT.

The hardware details of the Disk II Interface/RWTS system are discussed in some detail in Chapter 23. This is an unusual system in that some of the hardware control routines get involved in operating the drive machinery at a very tedious level. In many other drive/controller systems, the interface card is equipped with a floppy disk controller chip. The program which operates the chip simply passes the desired track and sector number to registers in the controller chip and then tells the chip to go into action. Many of these chips also respond to a simple one byte code by launching into a complete disk formatting procedure without any further assistance from the computer's microprocessor. The Disk II system, however, relies on time critical RWTS subroutines to actually send pulses to motors and manipulate the clocking system used in disk data storage.

Carrying Out a READ or a WRITE Command

When RWTS gets a READ command it turns on the drive motor to get the disk spinning. It can check with the Device Characteristics Table to find out if this is a Disk II drive, how long it takes this particular kind of drive to get up to speed, and how many step pulses it will have to send to move the head over by one track. Next, it gets the track number from the IOB and begins stepping the read/write head out to that track (see Chapter 23). RWTS tries to keep a record of where it left the head last, and how many stepping signals it has sent so that it can guess when it has arrived.

At this point it calls another one of its I/O routines, called RDADR, which reads the address bytes of the sectors spinning by the read/write head. The address of each sector includes the track number, so RWTS can find out at this point if it guessed wrong and adjust appropriately. If the track is correct, it watches for the particular one of the 16 sectors in the track which it is supposed to read (see Chapter 23, Figure 23.2).

In each sector, there are five or 10 synch bytes in what is called a gap between the address field and the data field. This means that after RWTS has read an address, it has about 150 to 200 microseconds (millionths of a second) to decide if this is the sector it is looking for. If it is, RWTS also has enough time to check with the IOB to find out if it is supposed to do a read. If so, it then starts capturing the incoming data and storing it in its decoding or postnibble buffers. The coding/decoding process is required by some of the physical limitations of the Disk II Interface system. After decoding, the data is moved to the sector buffer mentioned in the IOB, and RWTS hands control back to the File Manager.

A WRITE command is handled in almost exactly the same way, except that there is a pause after the track seek but before any address checking with RDADR. RWTS uses this pause to get the data for writing from the sector buffer and pass it through the coding or "prenibble" system. There is plenty of time to kill after a seek and before a RDADR is possible, because it takes about 20 milliseconds (thousandths of a second) for the read/write head to "settle" (stop shaking, etc.) after completing a step. This is enough time for the 6502 to carry out 5,000 instructions. With the head settled and the data prenibblized, RWTS starts reading sector addresses, and ultimately, does the writing.

When RWTS gets a FORMAT command, it hands control to its own INIT routine. This routine steps along track by track, and, in each track, writes in the addresses for each sector then uses RDADR to verify a successful format. When it has written in the address and synchronization fields for all 560 sectors, it finishes and the File Manager then uses a regular WRITE command to fill in the VTOC, and to write out 15 empty catalog sectors in track 17. Next, the File Manager goes about writing a complete copy of DOS into tracks 0 and 1, and then uses the standard SAVE command to get the Applesoft HELLO program written and cataloged on the disk.

The SEEK command is used only to position the read/write head over a desired track without actually reading or writing anything. Since track seeking is done automatically in all the other commands, SEEK is reserved for special purposes and not used very often.

Conversations with the File Manager

For the most part, in Apple DOS, all the File Manager's actions are controlled by the DOS Commands familiar to most Apple users (i.e., CATALOG, LOAD, RUN, INIT, etc.). These commands are usually issued from the Applesoft immediate mode in response to the square bracket prompt, or from within a program in a special PRINT statement (i.e., PRINT CHR$(4);"CATALOG").

When one of these commands is issued, it is first grabbed by the DOS Command Parser. This is a collection of routines which is responsible for recognizing your command, "parsing" out the various parameters such as slot, drive and record length (in random text file commands), and using this information to fill in the File Manager Parameter List at $B5BB (46,523). Once this parameter list has been filled in, the DOS Command Parser can call the File Manager into action with a JSR $03D6. This is yet another of those jumps through a page $03 vector, in this case pointing to the File Manager entry point at $AAFD (43,773).

Once the File Manager has been called with a properly prepared File Manager Parameter List, it goes about setting up a work area within one of its file buffers, and, when it is ready, it carries out the command by setting up the IOB and calling RWTS as many times as needed. A single command to the File Manager can cause it to set up the IOB and call RWTS hundreds of times.

With *Beneath Apple DOS* (by Worth and Lechner) in hand, it is possible for an assembly language programmer to skip the DOS Command Parser and work directly with the File Manager, or even directly with RWTS. However, Apple itself has only provided modest amounts of information on the RWTS interface, and essentially no information on the File Manager interface, so most folks have grown accustomed to dealing with DOS through its command parser.

Interception of Character I/O by the DOS Command Parser

Unlike CP/M and some other other operating systems, DOS and ProDOS do not have built-in routines for collecting character input from the keyboard and sending them to the screen for display. Those routines, under the supervision of GETLN, are provided by the Apple's Monitor ROM and are available even when DOS and ProDOS have not been loaded.

Incoming commands go straight to the Applesoft Interpreter, and Applesoft has absolutely no interface with the File Manager or the DOS Command Parser. In fact, if the Applesoft Interpreter receives a DOS command, it just won't recognize it, and it will respond with a Syntax Error statement. This is what happens if you, for instance, turn on the Apple without booting DOS, and then type: CATALOG.

The GETLN system for character I/O is discussed in detail in Chapter 33. GETLN is called by Applesoft to collect complete lines of characters terminated by a carriage return. It does its work with two hardware control routines; one for collecting bytes from the keyboard (KEYIN), and one for "echoing" them to the screen (COUT) so you can see what you're typing. The characters are also deposited in an Input Line Buffer where Applesoft can find them for analysis after you hit Return.

In order for DOS to detect commands typed at the keyboard or issued from within an Applesoft program, it must intervene in the Applesoft/GETLN system so that it may examine all commands before Applesoft sees them. DOS is able to recognize and act on them, and then effectively erase them from the Input Line Buffer so Applesoft never knows they were typed in.

DOS Changes the I/O Hooks

As described in the section on GETLN (see Chapter 33), the Apple has a convention for letting any device in any slot take the place of the keyboard in the KEYIN loop, and for permitting any device in any slot to take the place of COUT1 in the screen display loop. When GETLN is ready to fetch a new character, it effectively jumps through a location called KSW which can be changed to point to any input routine. Similarly, when GETLN is ready to echo the character to the screen, it jumps through CSW to whatever output routine it is pointing to. CSW can also be used directly by I/O routines other than GETLN (such as by the Applesoft PRINT routine) which need to send a character to the current output device, whether or not that device is video via COUT1.

When an Apple is turned on without DOS, KSW points to the KEYIN routine, and CSW points to the COUT1 routine. However, when DOS is loaded, both KSW and CSW are changed. Most of the action concerns the change in CSW. As you type characters at the keyboard, GETLN acts in its normal fashion. When it is ready to echo a character to the screen it puts the character in the 6502's accumulator and calls whatever routine is pointed to by CSW. When DOS is loaded, CSW points to the DOS Video Intercept Routine at $9EBD (40,637), and it is that routine that goes into action instead of COUT1.

Intercepting Commands Before Applesoft Sees Them

The Video Intercept Routine doesn't actually have anything to do with video, and, in fact, when it finishes its work, it usually puts the character back in the accumulator and calls COUT1 to do the actual display work. This is mentioned at this point only because this is a convenient place to peek into the stream of incoming characters to scan for DOS commands. One crucial point is that when you finish a line by hitting a carriage return, the carriage return gets sent to the screen handling routines via CSW *before* the Applesoft Interpreter is told that there is a completed line ready for it in the Input Line Buffer.

In the Immediate mode, when no Applesoft program is running, DOS simply waits for a carriage return to come along, and then, before allowing GETLN to finish its loop, it stops everything and examines the characters in the Input Line Buffer. If it recognizes a DOS command, it sets up the File Manager Parameter List and calls the File Manager to execute it. When the File Manager is done, the DOS Command Parser then effectively erases the contents of the Line Buffer, and then passes on the carriage return. All Applesoft ever sees is an empty line containing nothing but a carriage return.

When an Applesoft BASIC program is running, DOS commands are issued from the program in the form PRINT CHR$(4);"CATALOG". GETLN itself is not involved here, but the Applesoft Interpreter does try to use COUT1 to print the characters on the screen. BASIC programmers will recognize the "CHR$(4)" as a notation for the character "Control D." When Applesoft tries to print this statement, it will send the characters to the screen via CSW, and so they will be intercepted by DOS. When the Command Parser sees the Control-D, it knows that the following string will probably be a recognizable DOS command, so once the carriage return arrives, it will launch into a search of its command list.

The DOS commands MON and NOMON take effect here. Once the DOS Command Parser has seen the Control-D, it checks on your MON/NOMON instruction to decide whether to actually echo the characters to the screen, or just to handle the command internally and return control to the BASIC program when DOS is finished with the command.

There is one other crucial detail here. The DOS Command Parser will not check for a Control-D unless the previous character to pass through was a carriage return. Normally, this is no problem since DOS typically encounters a series of print statements from an Applesoft program, all terminated with an automatic carriage return. However, if you use the BASIC GET command, the user's keypress is echoed to the screen but BASIC sends no carriage return. If the next command is a DOS PRINT Control-D statement, the Command Parser will miss it. This can cause a frustrating bug in programs which want to ask "Hit any key to see a catalog of the disk" or some such. The remedy is to set D$ = CHR$(13) + CHR$(4). When you write PRINT D$;"CATALOG," Applesoft will send the CHR$(13) (which is equivalent to a carriage return) just before it sends the Control-D. There are some arcane occasions when this causes other problems, but it is a good short term solution to the GET/Control-D problem.

When the DOS OPEN and WRITE commands come along, the intercept is used so that when Applesoft's PRINT routine sends characters through CSW, they will end up being put into a data sector buffer instead of being put into display memory. If the MON O command is active, the character will then also be echoed to the screen via COUT1.

Intercepting Input Requests

DOS also changes the KSW input hook so that GETLN is forced to look to DOS every time it tries to collect a keypress. Normally, DOS just passes the request on along to KEYIN without interference. During a read from a text file on disk; however, DOS uses the intercept to pass

bytes from a data sector buffer to GETLN. In addition, DOS can use this intercept in its "EXEC" mode. An EXEC file is a text file which contains any or all of DOS commands, Applesoft program lines, and Applesoft immediate execution commands, such as LIST or RUN. When you type EXEC followed by the file name of the EXEC file, DOS fiddles with its keyboard intercept system. It opens the EXEC file and starts reading in the program lines and commands, character by character, just as if they were being typed from the keyboard. Each time GETLN tries to collect a keypress by calling KEYIN, DOS intervenes and hands back a character from the File Buffer.

The most common use for an EXEC file is to carry out a set of complex setup tasks before a standard BASIC program is loaded and run. This might include carrying out a set of POKEs to alter the Applesoft Interpreter's standard behavior in order to deal with very large programs, or it might ask the user a few preliminary questions before deciding which BASIC program to ultimately load and run. By making your HELLO file issue an EXEC command, you can make a "turnkey" system in which a casual user plops the disk in the drive, boots, and without any particular effort on their part causes a complex set of setup tasks to be carried out.

Chapter 35

Comparing Operating Systems: File Management

DOS 3.3, CP/M Plus, MS-DOS 2.0 and ProDOS

Although virtually all Apple II owners use DOS 3.3, nearly 20 percent of them also use some version of the CP/M operating system, a rapidly growing number are using ProDOS, and a powerful new implementation of MS-DOS 2.0 is attracting substantial enthusiasm. All of the fundamental elements of file management, space allocation, and disk I/O are essentially similar in the four operating systems, so a thorough reading of the preceeding full discussion of DOS 3.3 is also useful as an introduction to CP/M, ProDOS and MS-DOS 2.0.

To run CP/M, you must purchase a coprocessor board, usually with a Z-80 microprocessor (see Chapter 29), and should get an 80 column card as well. Hundreds of thousands of Apple owners have chosen to spend the extra $300 or $400 involved because CP/M offers strong attractions in terms of the power of the applications software it supports and in its flexibility with regard to adding high density disk drives. Both of these features reflect the internal structure of the operating system rather than any special power of the Z-80 microprocessor itself.

Very few Apple owners have shown much interest in purchasing 8088 based coprocessor boards to run MS-DOS 1.0 or CP/M 86. This makes good sense because these boards are often two or three times as expensive as Z-80 boards for CP/M, run almost exactly the same programs already available in DOS 3.3 or CP/M environments, offered negligible improvements in performance, and made no provision for disk compatibility with the IBM PC.

However, with the advent of MS-DOS 2.0, the appearance of powerful business programs using large amounts of memory (i.e., Lotus 1-2-3), and the release of Microsoft Windows (see Chapter 37), 8088 based systems have begun to offer very attractive features for businesses willing to pay a fair amount of money for the best performance currently available.

The Rana 8086/2 MS-DOS box (see Chapter 30) for the Apple II is faster and more powerful than an IBM PC, lets you continue to use all your Apple peripherals, and provides complete disk compatibility with standard IBM PC systems. Since this system has strong support from Apple Computer, from Microsoft (it was designed by the same engineer who made the famous Microsoft Z-80 Softcard for the Apple), and Lotus Development Corporation, the software outlook and modest pricing may make this one of the leading PC compatible systems.

Advantages of CP/M

The first reason that Apple owners have turned to Z-80 coprocessor boards in such large numbers is that several very high quality programs run on CP/M, notably the WordStar word processing program and the dBase II database system. The power and speed of these two programs reflects the powerful and flexible machine language interface available to the original programmers, and a fast and efficient means for rapidly loading "program overlays." An additional factor not related to the operating system is that many Apple DOS programmers worked strictly within the limits of the Apple's old 40 column screen, while CP/M programmers assumed their users would have high speed 80 column video systems.

The second reason that CP/M has been popular is that it adapts easily to hard disks, high density floppies, RamDisks, etc. Most Apple 6502 based software is tied strictly to the Disk II with its limiting and relatively obsolete 128K storage maximum. The real culprit is the RWTS routine. As explained in Chapter 23, the RWTS/Disk II Interface arrangement is a sort of open system which lets a programmer tamper with the fundamental hardware operation of the disk drive. All of the best Apple programmers have taken advantage of this to develop copy protection schemes in which their programs operate the Disk II Interface card in non-standard ways. If you try to use a different kind of drive, all of this copy protected, high quality software just won't work.

Another limitation of DOS along these lines is the Track/Sector approach to describing the file. The description allows one byte for the track and one byte for the sector, so you can have 256 tracks, each with 256 sectors. However, the newer hard disks offer up to 1,800 tracks each with 30 sectors, so you have to do major surgery on DOS just to help it describe your file. The alternative is to break the hard disk up into a large number of simulated smaller disk volumes, and then do minor surgery on DOS, but this is not ideal.

The New Role of ProDOS

ProDOS is a very important enhancement of the Apple because it offers a dramatic improvement in performance over DOS 3.3. It meets CP/M on its own ground, and does a better job than CP/M in several of CP/M's strongest areas. The first is a clean, efficient and straightforward machine language interface which improves on CP/M's BDOS call system. The second is that it will work with any sort of hard disk, RamDisk, high density floppy, etc., without requiring any modification of the operating system when a new kind of storage device is added.

It has also made the hardware interface substantially tamper proof to help prevent the unfortunate situation that developed with DOS 3.3 in which the best applications programmers shackled the Apple's growth in pursuit of the ultimate copy protection scheme. ProDOS arrives on the scene with a variety of strong software already available because it offers substantial compatibility with SOS on the Apple III.

ProDOS is an enhancement over CP/M because it includes an advanced "hierarchical" file structure which is oriented towards high speed management of hundreds of large files on a hard disk. It is also an enhancement in that it offers a simple menu driven user interface, so the non-programmer does not need to know any commands.

All of ProDOS, MS-DOS 2.0, and the new CP/M Plus include automatic time and date stamping of files, but they differ in their approaches to handling the greatly expanded amount of RAM memory now available for many microcomputers. ProDOS works in an environment in which

huge amounts of RAM can be bank switched into the 6502's address space for direct use by programs. CP/M Plus takes a very different tack in that it uses all extra RAM for operating system expansion, and for an enormously expanded file buffer system.

MS-DOS 2.0 for Very Large Storage and Multi-Tasking

The original version of MS-DOS 1.0 and PC-DOS 1.0 had just as many humbling limitations as DOS 3.3. This version of the operating system limited the PC to 160K drives and did not permit hard disks. The release of MS-DOS 1.1 allowed for double sided disks providing for 320K of storage, but this still did not make for an extremely attractive operating system.

MS-DOS 2.0 is a major overhaul of MS-DOS 1.1 which closely parallels the features which ProDOS brings to the 6502. Just as in ProDOS, MS-DOS 2.0 banishes the disk drive operation routines from the main body of the operating system. This means that any sort of mass storage device (hard disk, ram disk, etc.) can be conveniently added without major surgery to the operating system itself. In fact, MS-DOS 2.0 is even closer to ProDOS's elegant ancestor, Apple III SOS, in that the character I/O routines are also banished from the operating system.

Both MS-DOS 2.0 and ProDOS provide a hierarchical file structure for keeping track of large numbers of files on hard disks. The two systems differ in that ProDOS is designed for faster access to any given data block, but MS-DOS 2.0 can handle much larger hard disks.

The 16 megabyte maximum disk drive capacity permitted in ProDOS 1.0 is sufficient for the five, 10 and 15 megabyte hard disks in the popular Seagate 406 series (see Chapter 24) which include the Apple Profile and the IBM XT hard disk. ProDOS implements the Indexed Sequential Access Method (ISAM) (see below) for fast access to data in large files once the directory entry has been located, while MS-DOS 2.0 relies on a somewhat more clumsy extension of its allocation table system.

The newest 5¼ inch hard disks which provide up to 570 megabytes of storage, and the 1,000 megabyte optical disks overwhelm ProDOS's block allocation system. MS-DOS 2.0 has a byte oriented file system with a 32 bit address map, and this means that it can work conveniently with files up to 4,000 megabytes (four gigabytes) in size. To make this sort of thing practical, however, Microsoft or IBM may have to follow ProDOS's lead in setting up an ISAM system in future versions of MS-DOS.

The final area in which MS-DOS 2.0 excels over the initial release of ProDOS is in its memory management capabilities. The IBM PC applications environment seemed to be caught in a 64K rut until the advent of MS-DOS 2.0. The proliferation of multi-tasking and windowing systems for the IBM PC reflects the additional help provided by MS-DOS 2.0 in implementing these memory intensive user interfaces.

ProDOS 1.0 leaves Apple II programmers with the same problem that MS-DOS 1.0 presented. Large amounts of memory are available, via bank switching, but it is very difficult to take advantage of it. The introduction of the 65802 and 65816 microprocessors (see Chapter 30) can replace the complex bank switching setup with a large linear address space (up to 16 megabytes of RAM), but ProDOS does not anticipate this development.

The ORCA HOST extensions of ProDOS (from Hayden Software), for use with the 65816 and 65802, suggest a future path for expanding ProDOS to manage many megabytes of RAM memory in the Apple II itself, but MS-DOS 2.0 for the Rana 8086/2 represents the only

currently available system for general use. The 68000 based operating systems for the Apple Lisa and MacIntosh include very powerful memory management systems, but neither are available with 68000 coprocessor boards as described here.

Differences in File Description and Directory Structure

The four operating systems differ substantially in the organization of their directories. Relative to DOS 3.3, CP/M has a very simple directory structure. This eases the work of the CP/M file manager (called BDOS), but it leads to some serious inefficiencies with large directories. In contrast, the ProDOS directory structure is complex, powerful and efficient. As a result of the advanced design used, ProDOS can find any data sector in a 16 megabyte file by reading just three directory sectors, while it would take up to 1,000 directory sector reads for CP/M to find a data sector in a file of the same size. MS-DOS 2.0 needs a maximum of 50 sector reads for such a fill and also needs a modest amount of analysis after each read to trace its way through a potentially complex allocation chain.

A Review of the DOS 3.3 Directory System

In DOS, the file descriptive entry for each file includes the name of the file, a brief indication of its size, and the general nature of its contents, and a pointer to a special sector containing the first part of that file's track/sector list. The track/sector list tells the file manager which of the disks sectors have the data.

Each track/sector list sector has space for pointers to 122 data sectors, each of which hold 256 bytes of data. Thus, a single track/sector list sector describes 31K of data (see Chapter 34). For larger files, the first track/sector list sector can point to a second track/sector list sector to describe the next 31K of the file, and that one can point to a third if necessary and so on. This arrangement is called a "linked list;" once you have gotten to the first sector you can find your way to the third and so on.

The DOS file descriptive entries are themselves grouped into a linked list of catalog sectors. Each catalog sector has space for seven file descriptive entries and there is room for 15 catalog sectors on an Apple diskette, so there may be a maximum of $15 \times 7 = 105$ files in the directory. The anchor of the system is the Volume Table of Contents (VTOC) which is always placed in sector 0 of track 17. The VTOC points to the first catalog sector, and it also contains a bit map in which each of the disk's 560 sectors is assigned a position. When the file manager allocates a new sector to one of the files on the disk, it reads in the VTOC, marks the assignment in the bit map and rewrites the VTOC on the disk.

To locate a particular sector in a chosen file, the file manager first reads the VTOC sector, then it reads the first catalog sector and begins searching through the linked list of catalog sectors until it finds the file name it is looking for. From there it can find the first track/sector list sector for that file and start working its way through the linked list of track/sector list sectors until it finds the pointer to the one data sector it is looking for (see Figure 35.1).

CP/M Directory Entries and Records

In CP/M, the approximate equivalent of the DOS file descriptive entry is called a directory entry when it is on the disk, and it makes up most of a File Control Block (FCB) once it has been copied into memory. The directory entry has space for a short eight character file name

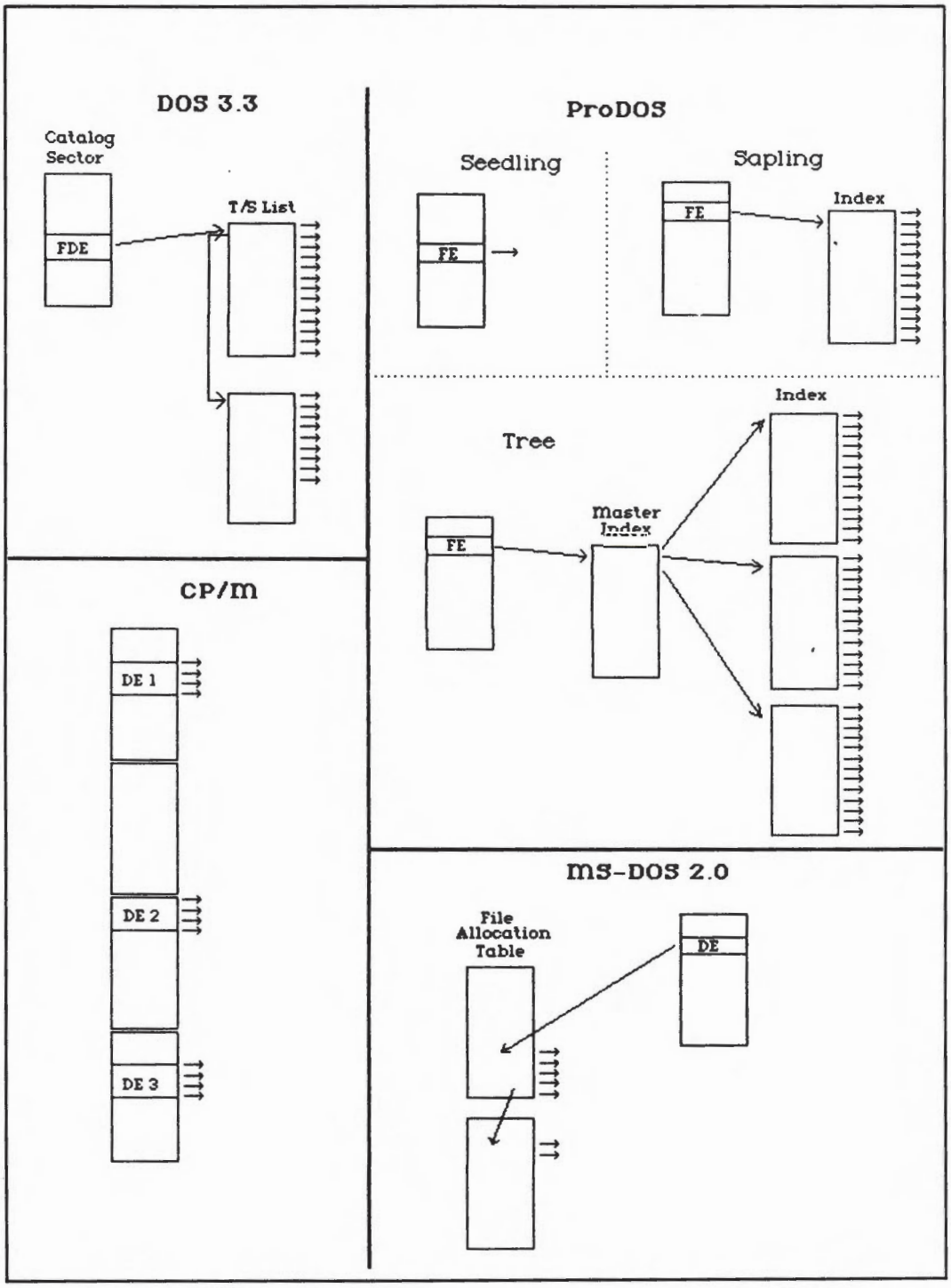


Fig. 35.1 File entry structure for DOS 3.3, CP/M, ProDOS and MS-DOS. FDE= file descriptive entry, DE = directory entry, and FE= file entry.

and an additional three character file type, and also has information on the size of the file. It differs from a DOS file descriptive entry in that the equivalent of the track/sector list is included in the directory entry itself.

In CP/M, the directory entry has no means of actually naming a track or sector. Rather, the CP/M Basic Disk Operating System (BDOS) deals mostly in Standard Records and Allocation Blocks. A CP/M Record plays much the same role as a DOS sector. It is the smallest amount of data that the BDOS can transfer between the disk and the buffers in RAM. However, in most CP/M systems it describes just 128 bytes. When CP/M is used with Apple disks, there are actually two 128 byte "logical" records in each 256 byte "physical" sector, but the BDOS is not at all perturbed by this fact.

Allocation Blocks and Extents

The BDOS filing system has a mental image of the disk surface in which it sees a single long string of up to 65,536 allocation blocks. Each of these blocks is 2K in length and contains eight records. When the BDOS wants to see a specific record, it asks for one of the allocation blocks by number (0 to 65,535) and then specifies one of the eight records within the chosen allocation block. For purposes of actually operating the disk drive, this must ultimately be translated into a request for a specific track and sector, but there is no sign of this at the level of the filing system.

Each directory entry has 16 bytes set aside as a data map. This data map is a list of allocation blocks, each of which is named by a unique two byte number (0 to 65,535). As a CP/M file grows, 2K allocation blocks are added to its data map much as T/S pairs are added to a DOS track/sector list. However, when eight of these allocation blocks have been written into the data map, that directory entry is full, there is no space for listing additional allocation blocks. Therefore, a single CP/M directory entry can describe just $8 \times 2K = 16K$ of data.

As soon as a file is larger than 16K, the BDOS must set up a new directory entry for the file. The first directory entry is said to describe the first "extent" of a file. Each extent contains 16K of data and has its own directory entry (see Figure 35.1).

The CP/M Disk Directory

A CP/M disk directory is made up of a simple long list of directory entries. There is no equivalent of the DOS VTOC, and no equivalent of the DOS track/sector list sectors. The directory entries are 32 bytes in length, so each 128K record of the disk directory has space for just four directory entries (see Figure 35.2).

Large files require several directory entries but there is no provision for grouping them all together. There are no pointers or links between the first, second, etc., directory entries which make up a complete file. When the BDOS needs to open a new extent for a file by setting up an additional directory entry, it simply goes out to the disk, scans along the single long list of directory entries, and, when it comes to the end, tacks on the new directory entry for the growing file. As a result, the directory entries of a large file can be scattered throughout the length of a long CP/M disk directory. During a read, when the BDOS is ready to look past the first 16K of a file, it must go on scanning the directory again until it encounters a repeat of the file name.

Enhancements of the Directory in CP/M Plus

The newest and most elaborate version of CP/M for the Z-80 processor is the banked version of CP/M Plus (also called CP/M 3.0). This requires more than 64K of RAM, and can handle

up to 16 banks of 48K each (see Chapter 32). The first extra bank, bank 0, is used for an expanded set of operating system features such as the support for time and date stamping of files, and the inclusion of passwords to limit access to files. In addition, CP/M Plus can use banks 2, 3, 4, etc., to improve the speed and efficiency of its disk directory searches.

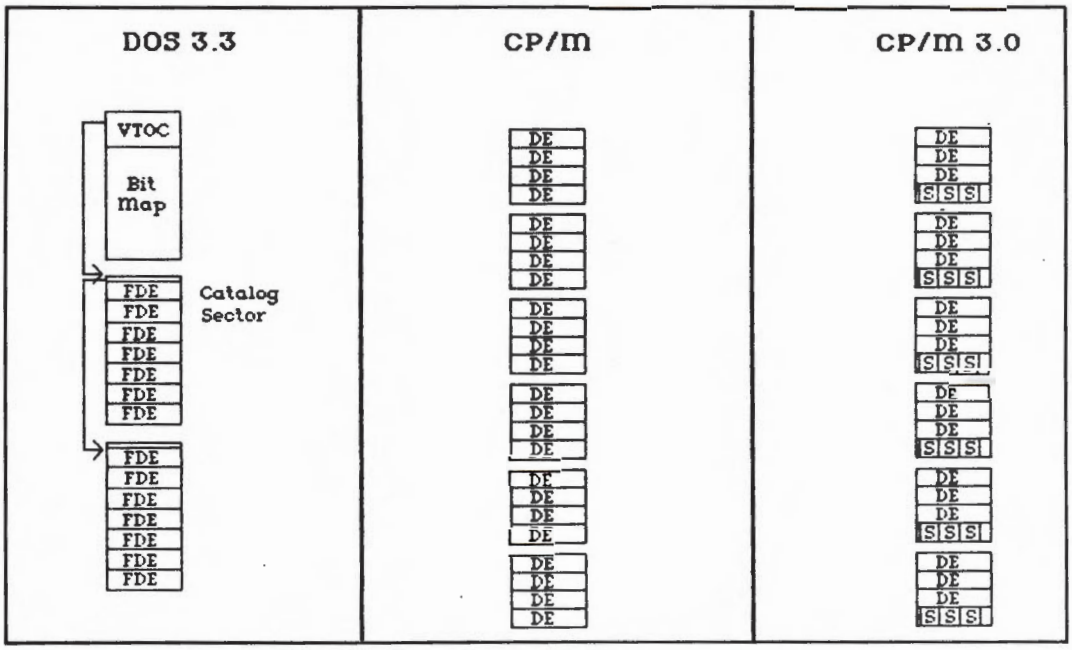


Fig. 35.2 Directory structure for DOS 3.3, CP/M and CP/M 3.0. VTOC= volume table of contents, DE= directory entry, and S=stamp (time and date).

In banked CP/M Plus (available with the Gold Card coprocessor from Digital Research), there are four kinds of directory entries. The standard directory entry which resembles the internal FCB is maintained, but there are also three new types. The key to the expanded system is the inclusion of a single "directory label" at the beginning of the disk directory which controls the use of the other two new types of entry. The first of these is the Stamp FCB (SFCB) which contains time and date information. In each record of the disk directory, there are three standard 32 byte directory entries, followed by three 10 byte SFCBs (see Figure 35.2) which describe the preceeding three directory entries. For password support, each file is also described by an Extended File Control Block (XFCB). This is similar in size and structure to a standard directory entry, but contains an encrypted password rather than a data map.

To speed directory searches, CP/M Plus uses two RAM based enhancements. Directory searching can be a serious problem for CP/M since each directory entry describes just 16K of the file, and only three directory entries fit in a record. To do a serial scan through the entire description of a four megabyte file, the BDOS will have to read over 250 records.

One old enhancement which is fully institutionalized in CP/M Plus is the "hash table." The BDOS builds a map of the directory so that once it has completed one full scan, it is able to head directly to the directory record it needs without doing a full serial scan in future accesses.

There is also a fairly elaborate buffer system for keeping copies of directory records in extra banks of RAM. The buffers and their assigned banks are set up on a fixed basis for each drive. As each directory record is read, a copy is left in one of the directory buffers. Future accesses of that directory record can be made to RAM rather than to disk. As the buffers fill up, the BDOS uses a "least recently used" priority system to decide which buffers to overwrite with new information.

ProDOS File Entries and Index Blocks

The basic file entry in ProDOS is fairly similar to a DOS file descriptive entry. The file name has been shortened to just 15 characters, and additional space has been made for date and time stamping. There is also an access field which the ProDOS block file manager can use to supervise backup of hard disks. Whenever a file is first written to, a "backup bit" in the access field is set to 1. Later, when it is time to back up the hard disk, ProDOS can selectively copy only those files which have been changed since the last backup. As the file is copied, its backup bit is reset to 0. CP/M offers a similar archive bit in the obscure position of the eighth bit of the third letter of the file type.

The major departure from DOS 3.3, however, is in the listing of the sectors in the file. ProDOS uses index blocks which are approximately equivalent to DOS 3.3 track/sector list sectors, but which are used in a somewhat different way.

The first major difference is that like CP/M (and like Apple Pascal 1.1), ProDOS does not refer to actual tracks and sectors, rather, it deals in "blocks," each containing 512 bytes. On the surface of the disk, the information on each block takes up two physical sectors. The two sectors in a block have an important physical relationship to each other which is described in some detail in Chapter 23.

Most disk drive systems cannot read two consecutive sectors because they must take so much time to process the first batch of data that they are not ready for another read when the next sector spins by a few thousandths of a second later. However, they are able to be ready for the sector immediately after the one they must miss. Whenever ProDOS does a read, it grabs one 256 byte sector to serve as the first half of a block, misses a sector, and then grabs a second 256 byte sector to make up the remainder of the full 512 byte block.

Branched Lists of Index Blocks

Another important difference which results from the use of 512 byte blocks is that each of these index blocks can describe much more data than a track/sector list sector. Each data block is described by a two byte number (0 to 65,535), and 256 of these two byte pointers can be packed into an index block. Since each data block holds 256 bytes of data, a single index block can describe 256 pointers × 512 bytes = 128K of data, as opposed to just 31K in a DOS track/sector list sector.

Much more important, however, is what happens when an index block fills up. ProDOS does not use a linked list of index blocks, rather, it uses a branched list. When it's time to set up a description of the 257th data block, ProDOS opens two new directory blocks. One of these is the second index block, but in addition, ProDOS sets up a "master index block" for that file. The file entry itself is also modified so that instead of pointing to the first index block, it now points to the new master index block. Within the master index block, two pointers are written in; one points to the original index block and the second points to the new index block (see Figure 35.1).

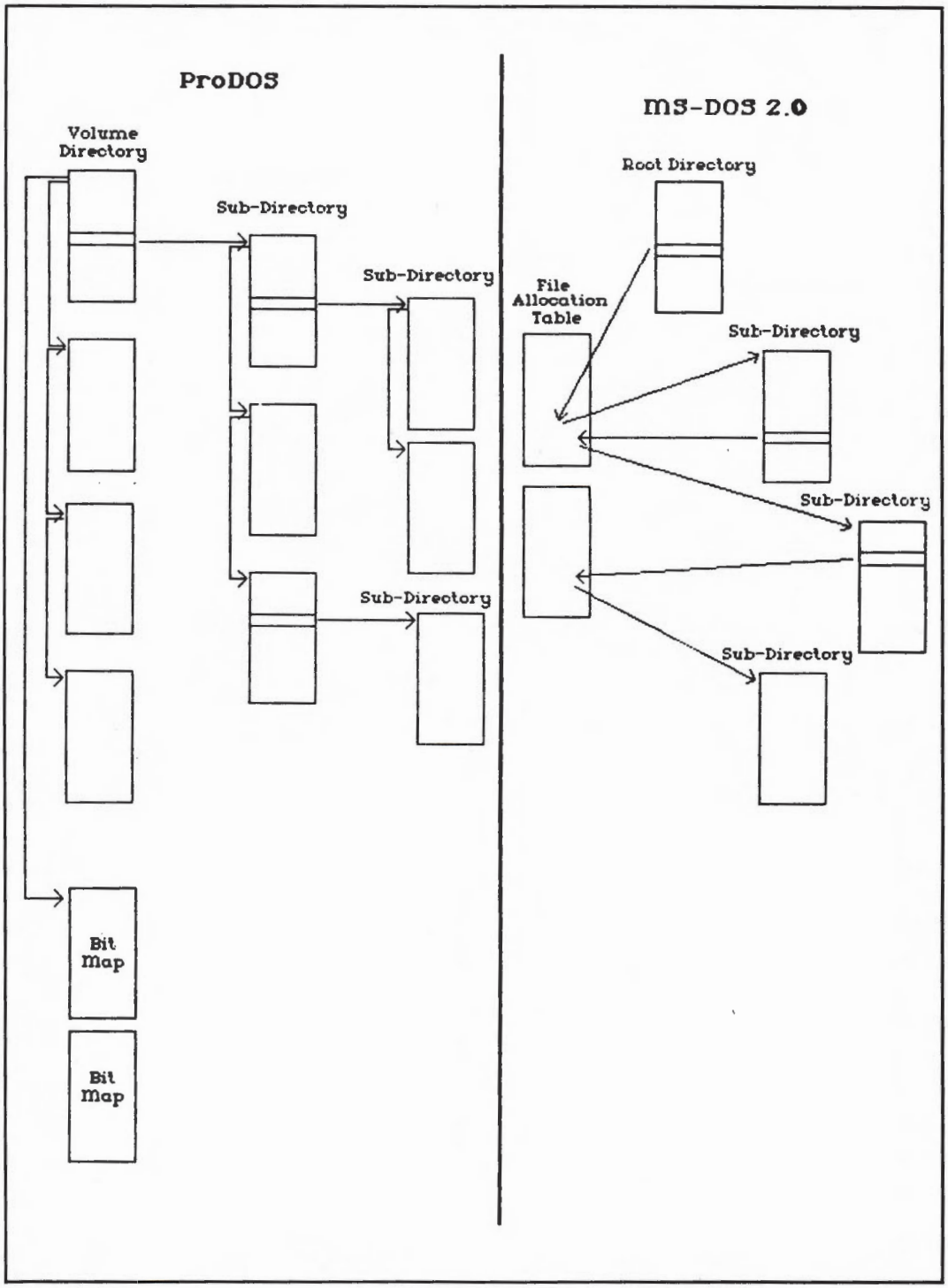


Fig. 35.3 Directory structure for ProDOS and MS-DOS 2.0.

As the file continues to grow, a third index block is set up, etc., up to 128 index blocks for a given file. If this were a linked list system, ProDOS might occasionally be forced to scan through all 128 of the index blocks to locate a desired data block number. In the branched list system, however, ProDOS is first sent to the master index block, and here it can find a pointer to any one of the index blocks it needs to look at.

The original file entry points to just one master index block. The master index block points to 128 index blocks and these each point to 256 data blocks. The file entry itself is compact and does not grow, but even as the file reaches 16 megabytes in size, ProDOS can locate any data block with one read of the master index block and one read of the selected index block.

This system of managing a large number of block addresses is called the Indexed Sequential Access Method (ISAM) and is one of several examples of the use of the best lessons of classic operating system design theory in the construction of ProDOS. As noted above, it could take over 1000 reads for CP/M to locate a single record in a 16 megabyte file.

Directory Blocks and the Volume Directory

Each ProDOS file entry (the name, size, type, pointer, etc.) is 39 bytes in length. These entries are collected in directory blocks, each with space for 13 File Entries (13 by 39 equals 507 bytes) with 5 bytes left over. ProDOS differs from both CP/M and Apple DOS 3.3, however, in that there is actually a complex organization of the directory blocks themselves.

The second, third, fourth and fifth blocks on every ProDOS disk make up that disk's "volume directory." These four blocks are laid out in a linked list arrangement. The first entry in the first of the four blocks is actually a "volume directory header" which stores the name and creation date for the whole disk volume, tells the total number of blocks on the disk, describes the format of the file entries, maintains a count of the number of active file entries in the volume directory, and points to the first block of the disk's bit map (see Figure 35.3).

ProDOS Hierarchical Subdirectories

The four blocks of the volume directory provide space for one header and 51 file entries. However, ProDOS permits a huge number of file entries to be stored on the disk through the use of "subdirectories." A ProDOS file entry can provide a name and pointer to a subdirectory, rather than to a standard data file. The 51 file entries in the volume directory can each point to a different subdirectory. Each subdirectory is a linked list of directory blocks with no limit on the number of blocks. Further, the file entries in the subdirectories can point to "sub-subdirectories" and so on, in a nested hierarchy which can be extended out to several levels of nesting.

The nested hierarchy system for ProDOS directory blocks is extremely efficient for work by the ProDOS block file manager, and it is also very helpful to users. Those of you who have had the good fortune to use hard disks or megabyte floppies will probably be well aware of how difficult things get once you've got 150 or 200 files on a disk. Your task in scanning the directory listing is extremely frustrating. First, the whole thing won't fit on the screen at one time, but worse, it just simply becomes impossible to keep track of what's out there by picking two or three entries out of a huge list.

In ProDOS, you organize your files into nested subdirectories and therefore avoid most of these problems. In a typical setup, you would keep the volume directory fairly small with one entry for all your text files, one entry for all your Applesoft programs, and one for all your commercial software, each pointing to its own subdirectory. Within the Applesoft subdirectory

you might have names for each of your ongoing projects. You would finally select the one project you were interested in and get on out to its own subdirectory. There you would find the actual file names with date and time stamps for the various recent versions of your program.

Pathnames, Seedlings, Saplings and Trees

The header of the volume directory stores a name for the disk, and the first, or "key," block of each subdirectory also has a name. Thus, in the example described in the preceeding paragraph, your fully named description of the "path" to the file might be /Maindisk/Apple-soft.Progs/Graphics/Current.Vers. This is called a ProDOS "pathname," and includes the name of a volume, two levels of subdirectories, and an actual file name. The limit on nesting of subdirectories is that the entire pathname must not exceed 64 characters.

There are four kinds of file entries in the various parts of a ProDOS directory. The ones that point to subdirectories are simply called "subdirectory file entries," but there are three kinds of file entries for pointing to actual data blocks.

When there is just one block of data in a file, the file entry points directly to it, and that file entry is said to be a "seedling file entry." As described above, once a second data block is added, there must be an index block. The file entry points to the index block and the index block has pointers to up to 256 data blocks. Those file entries which point to index blocks are called "sapling file entries." Once there are more than 256 data blocks, a second index block is created as well as a master index block. The file entry now points to the master index block and is called a "tree file entry" (see Figure 35.1).

The Mark, Reference Number and Bit Map

ProDOS can describe each byte in a file with a three byte structure called a mark. The first seven bits of the mark tell which of the 127 pointers in the master index block is involved, the next eight bits select one of the 256 pointers in the chosen index block, and the final nine bits of the mark select one byte from within the 512 bytes of data in the data block. This three byte mark structure serves to create an address space for the file which effectively assigns a 24 bit address to each byte in the file much as MS-DOS 2.0 uses a four byte read/write address pointer to set up a 32 bit file address.

Once you have typed in a full pathname for ProDOS to use, it stores that name in a list in RAM and assigns it a reference number. During most data manipulations, ProDOS leaves the full pathname in the list and uses the one byte reference number to refer to the full pathname. The reference number is comparable to the file handle in MS-DOS 2.0 (see below), except that only one reference number can be assigned to a file and these numbers can refer only to files and not to devices as well. Together, the reference number and the mark provide a compact four byte description of exactly where the ProDOS block file manager can find any individual byte of a truly enormous array of huge files.

Like DOS 3.3, ProDOS maintains a bit map on the disk which describes all of the blocks. ProDOS bit maps are stored in their own separate blocks, beginning with block number 6. Each bit map block can describe 4,096 blocks on the disk. When the block file manager needs a new block, it scans the bit map for the first available block, sets that bit in the map, and then uses the mark and an end of file value to determine where to write the new block number in the directory.

MS-DOS 2.0 Directory Entries and the File Allocation Table

The full description of a MS-DOS file is organized quite a bit differently than a DOS 3.3, CP/M, or ProDOS file. The directory entry itself most closely resembles a simplified version of a ProDOS file entry. It has space for a 10 byte file name and extent, one attribute byte for use in archival backup (see above) and in defining read/write status, a four byte description of total file size, and four bytes for keeping track of date and time. The date and time represents the last use of the file, but, unlike ProDOS, there is no provision for retaining the original create date as well.

Two more bytes in the MS-DOS directory entry are used to describe the physical location on the disk where the file's data begins. The elemental unit for disk space allocation in MS-DOS is called a "Cluster," so these two bytes in the directory entry are called the "Starting Cluster Number."

The MS-DOS Cluster system is the unique aspect of the file description. These Clusters differ from allocation units in the other three operating systems in that their size can be changed to best adapt to whatever kind of physical storage media is being used. Further, the organization of Cluster lists within a mapped File Allocation Table (FAT) is unique, although a bit inefficient.

Clusters and Sectors

As in the other operating systems, the actual physical unit of storage on disks is the sector, although MS-DOS sectors are usually 512 bytes rather than the 256 bytes used in other systems. In MS-DOS 1.0, the single side of a floppy had 40 tracks, and each track held eight of these 512 byte sectors; thus the original standard IBM PC drive held 160K of data. The 40 $\times$ 8 = 320 sectors on the disk were each assigned a number by which they could be addressed, and a FAT was set up on one of the sectors on the disk in order to keep track of the sector numbers.

The original design of the FAT sector called for it to serve both as a map of allocated sectors on the disk and as a list of the sector numbers. A one byte sector number could only describe 256 sectors so was not sufficient. A two byte sector number allowed for over 65,000 sectors, but a list of 320 of these two byte numbers would be 320 $\times$ 2 = 640 bytes long and so wouldn't fit into a single FAT sector. All of this led Tim Patterson (the grandfather of MS-DOS) to use a 12 bit number (that's right folks, one and a half bytes) to represent each sector. A list of 320 of these 1.5 byte numbers fits neatly into a single FAT sector (320 $\times$ 1.5 = 480).

The Cluster becomes important with the arrival of MS-DOS 1.1 for double sided disks; there are now twice as many sectors to keep track of. One solution would have been to add a second FAT sector, but instead, the Cluster came into use. When MS-DOS 1.1 is using a double sided drive, it defines a Cluster as equal to two contiguous sectors, but when it is using a single sided drive, it defines a Cluster as being equivalent to one standard 512 byte sector.

In MS-DOS 1.1 and MS-DOS 2.0, the 12 bit numbers in the FAT are cluster numbers rather than sector numbers. A Cluster is 512 bytes for single sided floppies, 1,024 bytes for double sided floppies, and is variable in size for hard disks. Although the use of a 12 bit number must have seemed to make good sense in 1979, it is certainly problematic now. It allows for only 4,096 different cluster numbers, and this runs against the grain of MS- DOS 2.0's capabilities for managing very large files. Simple math shows that 4,096 clusters, each containing 1,024

bytes, gets you a total disk capacity of just four megabytes. To take advantage of the full four gigabyte addressing capability of MS-DOS 2.0, each cluster must allocate a full megabyte (2,048 sectors) at a time, even if a given file is only a few bytes long.

One further wrinkle in the floppy disk Cluster system is that MS-DOS 2.0 allows the option of squeezing nine sectors into a track. When this nine sector option is active, a single sided floppy disk can hold 180K of data and a double sided disk gets you 360K. The matching of sector numbers to cluster numbers is also altered between MS-DOS 1.1 and MS-DOS 2.0. In version 1.1, the sectors on the first side are numbered zero through 319, and the numbers on the second side continue upwards from there. In version 2.0, however, the numbering scheme helps simplify the efficient use of the two sided drive by putting sectors zero through eight on the first side of track zero, sectors nine through 17 on the second side of track zero and so on. For further nuances of relative sector numbering, etc. see *The IBM PC-DOS Handbook* by Richard Allen King (Sybex).

Chained List of Clusters in the File Allocation Table

The File Allocation Table (FAT) can be represented as a stack of 12 bit fields for storing Cluster numbers. This table is a map of the disk surface in that each of the fields is matched to one cluster. The first field is for Cluster 0, the second for Cluster 1, etc. These map assignments are true no matter what the contents of the fields.

When a Cluster is not in use, the number $000 is stored in its field in the FAT. This permits MS-DOS to use the FAT much as DOS 3.3 and ProDOS use their bit maps (see above). However, the MS-DOS FAT is able to simultaneously play the role of the Track/Sector List in DOS 3.3 because of the ability to store file information in the FAT fields. In addition, MS-DOS takes advantage of this ability by putting an $FF7 into the FAT field of any Cluster it knows to be damaged or unreadable thus removing that Cluster from the allocation pool.

When an MS-DOS file is first opened, a Directory Entry is set up, and it is assigned its first Cluster on the disk. The 12 bit number for that first Cluster is stored in the Starting Cluster Number bytes in the Directory Entry itself, and the field in the FAT which represents that Cluster is also marked.

Initially, that first assigned field in the FAT for that file is labeled $FFx (where x is between $8 and $F) which is a notation for End of File. When MS-DOS is ready to assign a second Cluster to the file, it scans the FAT to find an available Cluster and then writes the number for that cluster over the $FFx it had previously recorded in the first assigned field. It can then put the $FFx in the field for the second assigned cluster.

Every Cluster which contains active data is noted in the FAT with a number in its matched FAT field. If that Cluster is the last Cluster in the file, the field contains $FFx. Otherwise, the field contains the number of the next Cluster in which that file's data is stored. A complete file description therefore begins with the Starting Cluster Number in the Directory Entry, and then threads though a chained list of Cluster numbers in the FAT (see Figure 35.1).

Hierarchical Directory Organization in MS-DOS 2.0

An MS-DOS directory is made up of Directory Sectors and FAT Sectors. In MS-DOS 1.1, these sectors are all placed in track 0, the outermost track, in a standard sequence. The first sector in track 0 is reserved for the bootstrap program, so the directory begins with the second sector. The FAT Sector is placed first, followed immediately by an exact duplicate copy of the principal FAT Sector. MS-DOS makes extremely heavy use of this one sector, so the inclusion of a duplicate copy dodges the disaster which would occur when the main FAT sector gets damaged by wear and tear.

The two FAT Sectors are followed immediately by four Directory Sectors. Since each directory entry is 32 bytes in length, 16 entries fit in each Directory Sector, and there is space for a total of 64 directory entries. When a double sided disk is in use, there are still just two FAT Sectors, but MS-DOS 1.1 allows for seven Directory Sectors, so there is space for $7 \times 16 =$ 112 Directory Entries.

An MS-DOS 2.0 directory is different from an MS-DOS 1.1 directory in two ways. First, it must allow for a large enough map of FAT fields to describe all the Clusters on a nine sector floppy diskette. Therefore, there are two primary FAT Sectors to accommodate the enlarged map, followed by two copies, for a total of four FAT Sectors. There are still four Directory Sectors for single sided diskettes, and seven Directory Sectors for double sided diskettes. For Hard disk support, there may be up to 12 primary FAT sectors and 12 FAT copy sectors, but they are still followed by just seven Directory Sectors.

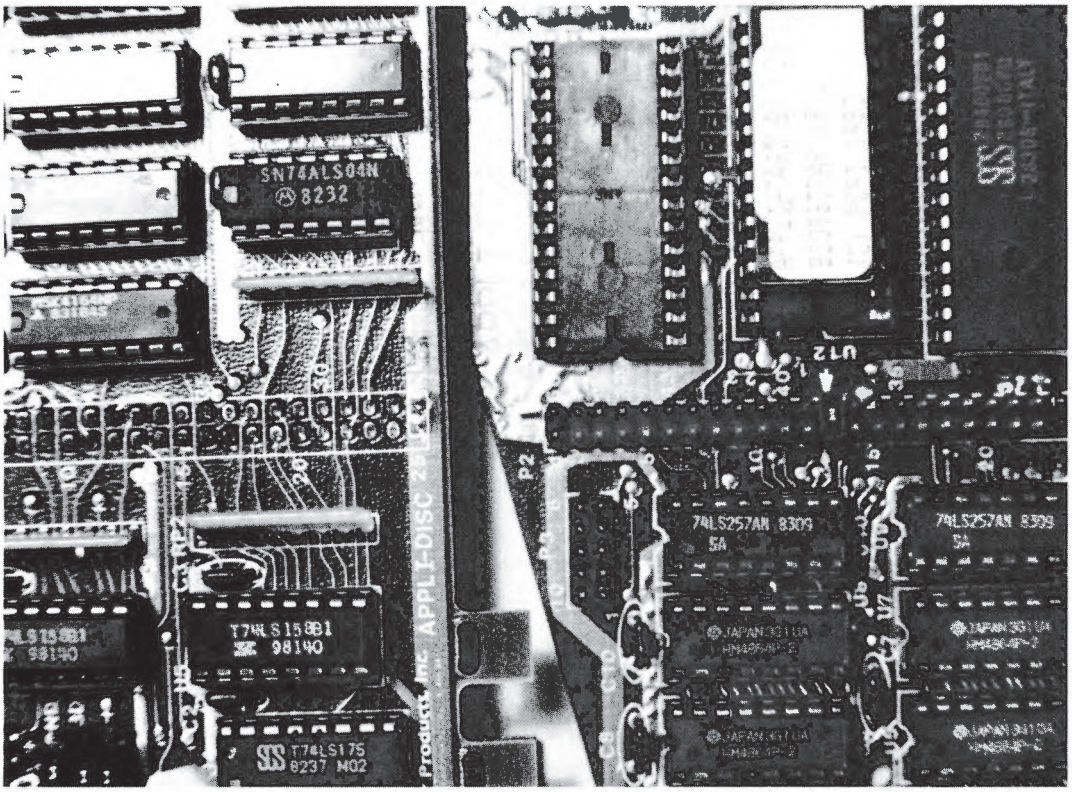
The second and crucial difference in MS-DOS 2.0 is that the seven Directory Sectors immediately following the FATs are designated as the disk's Root Directory, and any number of additional Directory Sectors may be added later, scattered about the surface of the disk. The primary function of the Root Directory is to keep track of the additional scattered Directory Sectors, and the whole system is tied together into a UNIX-like hierarchical directory structure which resembles the ProDOS directory in a number of ways.

Hierarchical directories are extremely important for hard disk oriented operating systems, because it is unreasonable for a user or for MS-DOS itself to attempt to stay oriented and efficient when confronted with an undistinguished list of hundreds of file names. This concept is described in some detail in the ProDOS section above.

The physical implementation of the hierarchical directory system is a bit more clumsy than than the ProDOS system since it uses chained lists of Cluster numbers in the FAT (see Figure 35.3) rather than independent linked lists and keys as in ProDOS. The result of this structure is to make the system a little more sluggish and a little less space efficient (you may have to reserve an entire 8K Cluster for one or two 32 byte directory entries), but the appearance to the user is almost identical to the ProDOS system.

Files are addressed with "Paths" made up of the names of a series of subdirectories followed by a file name, each separated by a slash (i.e. B:/Workdisk.Myp/Graphics.new/Basic.prg). Unlike ProDOS, the Path does not include a name for the volume, but requires a CP/M-like drive identifier (i.e., A: or B: etc.). Paths may be up to 63 characters in length, and a programmer can refer to a path string with a 16 bit number called a File Handle when doing machine language calls to the operating system.

Within the file itself, individual bytes are selected with a 32 bit identifying number called the "Read/Write Pointer Address" and the blocks and records used by MS-DOS 1.1 are completely abandoned. Therefore, the two byte file handle followed by the four byte Read/Write Pointer Address can select any byte within a four gigabyte file, and any file from among 65,536 path strings. This sort of feat is appropriate for a huge modem oriented database system well outside MS-DOS's single user type of environment. It reflects a trend toward compatibility with Microsoft's multi-user Xenix operating system rather than any anticipated real demands on MS-DOS.


SN74ALS04N
8232
T74LS158B1
98140
T74LS175
8237 M02
74LS257AN 8309
5A
74LS257AN 8309
5A

Chapter 36

Comparing Operating Systems: I/O Services

Buffers for Data Transfers in DOS, CP/M, ProDOS and MS-DOS

As outlined in the section on DOS buffers (see Chapter 34), a data transfer involves creating an effective link between a unit of information on the disk and an equal size unit in RAM memory, and then copying from one to the other. In DOS 3.3, Sector Buffers are 256 bytes, CP/M uses Record Buffers of 128 bytes, ProDOS needs block sized buffers of 512 bytes, and MS-DOS 2.0 uses a Data Transfer Area of up to 64K bytes. However, in addition to the differences in size, there are substantial differences in the organization of the buffer system in the four operating systems.

Fixed DOS File Buffers

In DOS 3.3, one buffer is set aside for the VTOC, one for the active Catalog Sector, and then three larger File Buffers are used for all data transfers. Each File Buffer has 256 bytes for the active Track/Sector List Sector, 256 bytes for the current data sector, and an additional 73 bytes for various housekeeping information associated with the file. The three 595 byte File Buffers are placed in RAM memory between $9600 and $9CF9 (38,400 and 40,185; see Chapter 34, Figure 34.1).

All DOS data transfers, for programs and for text, are done via the File Buffers in high memory. After each incoming sector arrives, the data is moved to its intended location in a comparatively slow byte by byte relocation loop.

Movable CP/M Record Buffers

CP/M has a much more flexible buffer system which requires much less space in memory, and which provides for very fast loading of programs. When DOS 3.3 wants to start a data transfer, it must make space for and read the VTOC, a Catalog Sector, and a Track/Sector List in addition to the data, all four of which require 256 bytes. CP/M has no equivalent of the VTOC, and a single 32 byte Directory Entry plays the role of both the Catalog Sector and the Track/Sector List. CP/M simply scans the directory until it finds the record which contains the desired Directory Entry. The BDOS then sets up a File Control Block (FCB) which includes

the 32 bytes of the Directory Entry and four additional bytes to help keep track of the current position within the file. Of course, this is only good for 16K of data, after which the files second Directory Entry must be loaded, etc.

The data itself can be sent to any free 128 byte area in RAM memory. There is a default data buffer in the zero page ($0080 to $00FF, decimal 128 to 255) which is always protected from the running program; however, most CP/M based commercial software makes extensive use of the flexibility and small size of the data buffer system. As discussed earlier (see Chapter 35), it is often preferable to load programs into memory at the position where they will actually be used. The buffer is set up at the starting location for the first incoming record, and then advanced 128 bytes in memory for the next incoming record, etc., until the entire program is loaded in place. This is far faster than the method used in DOS.

CP/M Overlays and CP/M Plus Banked Buffers

The CP/M BDOS loads all .COM files in this way, starting at location $0100 and working its way upward, but there is another important variation on this. One of the reasons why WordStar is such a powerful word processing program is that it is actually over 80K in length, while most DOS word processing programs try to squeeze into 20 or 30K of Apple RAM.

WordStar is able to work perfectly well in a 64K Apple II because the entire program is never completely in memory at one time. There is a core portion of the program which is always present, but as you request additional functions from the program, the core portion calls out to the disk and brings in the portion needed. When you need to do something else, the core calls back out to disk, locates a new section, and overwrites the section it brought in previously. This is called an overlay system and it is also used in dBase II. The overlay loads are very fast, so the user experiences WordStar as if it were a huge 80K machine language word processing program. Hence its reputation as a diverse program which can do almost anything you might wish to have done without ever exiting from within your text.

The Banked Version of CP/M Plus offers an additional enhancement to the standard CP/M data record buffer system. The improvement is quite similar to the CP/M Plus directory buffers (see Chapter 35). All data record reads can be copied to extra banks of RAM where the BDOS will keep track of a number of extra data record buffers. Future accesses to the same data record can be made to RAM instead of to disk. Once the buffers are all full, the BDOS uses a Least Recently Used priority system to decide which ones to overwrite.

ProDOS Buffer Options

The ProDOS buffer system for Applesoft programs and text files is similar to the DOS 3.3 system, but ProDOS provides additional options. There can be up to eight active File Buffers in ProDOS, each storing 512 bytes of data and a 512 byte Index. When all eight of these are in use, the File Buffers take up 8K in high RAM. ProDOS does not permanently reserve this much space, but grabs it in 1K increments as needed. This means that, unlike DOS 3.3, there is no permanent and secure HIMEM value when ProDOS is running.

There is a second method used for loading machine language .system files. These are automatically loaded beginning at $2000 (8,192) and then moved to their final destination. It is also possible to do the equivalent of a DOS BLOAD of an Applesoft program or a text file, thus providing better control over the loading process and a substantial improvement in speed relative to DOS. An assembly language programmer can specify a starting location for the load, and any amount of data which will fit in free memory.

The MS-DOS Data Transfer Address

The data buffer system used in MS-DOS is virtually identical to the CP/M system. In MS-DOS 1.1, this was explicitly the case although the relation is a little less obvious in MS-DOS 2.0. The important differences have to do with the use of 8088 Segment Registers (see Chapter 30) to divide memory space into four active 64K segments. The beginning of the CP/M zero page is equivalent to the beginning of the program segment in MS-DOS. Just as with the CP/M zero page, the first $100 (decimal 256) bytes of memory in the program segment are reserved by MS-DOS as a Program Segment Prefix. The second 128 bytes of this area (addresses CS:$0080 to CS:$00FF) are used as a default file buffer for a 128 byte data transfer. MS-DOS 1.1 uses Records with a default size of 128 bytes, hence it duplicates CP/M exactly.

The address used to describe the location of the buffer is a bit more tricky than in CP/M because of the use of Segment addresses. MS-DOS transfers data into the Data Segment rather than into the Code Segment. In the default condition, then, the Data Segment is set to begin at byte $0080 of the Code Segment. The substantial flexibility of MS-DOS data buffers arises from this opportunity to set the beginning of the Data Segment to any available area in memory. Further, MS-DOS 1.1 allows any Record size from one byte to 64K bytes.

The Data Segment Register can be set to point to a wide open free area quite separate from the location of the program itself. However, the segment architecture of the 8088 does impose a constraint in that MS-DOS will not load data across a segment boundary. This means that the maximum buffer size is 64K bytes. This also means that MS-DOS cannot do full cylinder reads from very high capacity hard disks, but is more than adequate for accepting the 32K bytes contained in a cylinder of the 10 megabyte hard disk supplied with the IBM XT.

In MS-DOS 2.0 there are no Records, and the CP/M-like FCB system of MS-DOS 1.1 is not apparent to the programmer. The buffer address is set up and a request is made for any number of bytes from anywhere within the file. The limit imposed by the 8088 segment boundaries still applies though, so the largest single read is for 64K bytes.

Different Approaches to Operating the Hardware

Apple DOS 3.3 describes its disk storage locations directly in terms of the physical tracks and sectors, and the RWTS routine which actually moves the data is designed to work almost exclusively with 128K Disk II drives. DOS 3.3 is therefore deeply rooted in a single kind of relatively low capacity disk storage. CP/M is designed so that an assembly language programmer can alter it to operate with nearly any kind of disk storage device. ProDOS is actually device independent. The simple act of installing an appropriate interface card in an accessory slot is sufficient to get ProDOS to work with any kind of drive.

To get a sector of data, the DOS 3.3 File Manager begins by picking the appropriate track/sector pair from the Track/Sector List in one of its buffers. It stores the track number and the sector number in the I/O Control Block (IOB) along with a little bit of supplementary operation, and then calls RWTS to do the transfer (see Chapter 34). The RWTS section of DOS has some modest responsibilities for manipulating the data as it passes through, but its chief responsibility is to select and run the appropriate one of its six major hardware control subroutines. These subroutines are intimately involved in the operation of the Disk II Interface card (or IWM in the //c) and the drive, at the level of sending timed pulses to the stepper motor and

interacting with the data coding clock loops (see Chapter 23). This is why you cannot just attach some completely different kind of disk equipment.

Operation of the CP/M BIOS

Although the BDOS portion of CP/M (see Chapter 35) is distributed in a standardized form from Digital Research, the BIOS portion which they send out is only a skeleton which must be customized by the computer manufacturer (i.e., Microsoft, ALS, PCPI, etc. for Apple cards). The BDOS views its data as groups of 128 byte Records in 2K Allocation blocks; however, when it actually requests a read or a write, it pauses to determine exactly which physical track and sector corresponds to the record of interest.

The BIOS is expected to contain a set of tables which describe the physical attributes of each of the drives in the system. To fetch a record, the BDOS first collects the appropriate information from one of the tables in the BIOS, calculates the physical track and sector, and stores the result in a parameter exchange location.

The BIOS is also supposed to include a collection of hardware control routines, with a complete set available for each different kind of drive in the system. Once the BDOS has calculated and stored the values for the actual track and sector, it selects the appropriate hardware control routine from within the BIOS library and calls it.

The full sequence of events is as follows. The user or application program specifies a drive such as A: and the BDOS responds by calling the SELDSK routine in the BIOS. The BIOS responds by going to its Drive Table and finding the address of the Extended Disk Parameter Header (XDPH) which fully describes that drive and the location of that drives hardware control routines. The BDOS then uses data in a portion of the XDPH to perform calculations which convert an absolute record number into a physical track and sector value. The BDOS may also call a BIOS routine called SECTRAN which handles skewing (see Chapter 23, Table 23.1).

With everything ready, the BDOS calls SETTRK to store the track number, SETSEC to store the sector number, and SETDMA to specify the location of the buffer in memory. In a Banked CP/M Plus system it can also use SETBANK to choose which bank of RAM will receive the data. Finally it calls READ or WRITE, and the BIOS uses the data from the XDPH to call the correct hardware control routines. The actual read and write subroutines used in the standard version of Microsoft CP/M for the Apple are essentially identical to the ones used by RWTS, and the Microsoft card actually turns control over to the 6502 while the RWTS-like hardware routines are run. Control comes back to the Z-80 at the end and the data is picked up from a common buffer location.

The XDPH also contains information used by the BDOS in managing the record allocation process. Recall that unlike DOS 3.3 and ProDOS, there is no bit map on CP/M diskettes. However, the BDOS does attempt to maintain a bit map of Allocation Blocks in RAM, and this happens to be called an Allocation Vector (ALV). The XDPH points to the address where the BDOS should set up the ALV for a given drive and it also tells the BDOS which blocks should be reserved for the directory records and other housekeeping chores.

When CP/M first uses a drive, it sets up an ALV for it and then does a complete read of the directory to mark all the used blocks as bits in the ALV. If you change disks without letting the BDOS first set up a new ALV for the disk you just slipped in, you risk a disastrous event of overwriting data.

Why PIP Sometimes Gets an R/O Error

To prevent this, one might have the BDOS redo the ALV every time it looked at the disk, but that would be very time consuming since it requires a complete read of the directory. Rather, when the BDOS first sets up a disk's ALV, it also sets up a Check Vector which is nothing more than an image of the first part of the disk's directory. Before every access, it compares this Check Vector with the beginning of the directory and if they are not identical, it knows that you have changed the disk, that its ALV is useless, and that it could get into big trouble fast. It responds by declaring the disk to be read only, refusing to write anything, and issuing an error statement, the notorious R/O Error.

The XDPH specifies how far the BDOS should look into the directory in setting up and comparing its Check Vector. The further it has to look every time, the longer it will take to do reads and writes. However, if the first few entries on two directories are identical, a short Check Vector might not detect the disk change. This finally brings us to the annoying behavior of PIP.

Apple DOS users are accustomed to freely popping disks in and out of drives (except Screen Writer II users; see Chapter 34) and many try to continue with this when they start using CP/M. In fact, you must *always* type Control-C at the system level when you change disks unless you are specifically told otherwise by some program. In the normal course of things, people who don't follow this rule strictly find that PIP works fine sometimes but gives R/O errors at other times. And there is nothing like an intermittent problem to drive a computer user bonkers. The reason PIP works sometimes without the Control-C is just that the Check Vector does not always extend far enough into the directory to detect a change on two disks which originally stored the same files.

The ProDOS Disk Driver System

ProDOS differs sharply from both DOS 3.3 and CP/M in that it does not have any hardware control routines at all. There is no part of ProDOS which can be considered equivalent to RWTS or the CP/M BIOS. Further, data blocks are always treated as data blocks, and no part of ProDOS ever deals directly with tracks and sectors.

When ProDOS wishes to carry out a data transfer, it loads four pieces of information into zero page locations. The first of these is a code at $0042 for one of four actions: a read, a write, formatting, or a report on the drive's status. A second code, called a Unit Number, is stored at $0043 and it specifies the particular slot and drive to be used. It sets $0044 and $0045 to point to a buffer in RAM, and it puts the desired block number in locations $0046 and $0047 (these six addresses are decimal 66 to 71). With these locations set up for command, drive, buffer address and block, it calls some disk driver routine that is not part of ProDOS to actually do the transfer.

When ProDOS is first loaded into memory, it sets up a Global Page beginning at $BF00 (48,896) in which it stores a variety of special system variables. Within the global page, between $BF10 and $BF30, ProDOS sets up a table of Disk Device Driver Vectors which is used whenever it needs to call a disk driver routine. There are 16 positions in the table, one for each of the two drives in each of the eight peripheral slots.

One of the first things ProDOS does when it is booted is to systematically attempt to read the Peripheral Card ROM ($CN00 ROM; see Chapter 22) on all installed peripheral cards. These ROMs are expected to have "signature bytes" which tell ProDOS what kind of card they are

on. If a card is a disk interface card, the ROM is supposed to tell how many blocks fit on a disk and whether the medium is fixed or removable.

Most importantly for the ProDOS device independent system, it is also supposed to provide ProDOS with the entry address for calling a complete set of peripheral management and hardware control routines collectively called a "Disk Device Driver." There is usually plenty of room for these routines in the 2K of Expansion ROM space on the interface card. These routines must be able to examine the information at locations $0042 through $0047 (see above), translate the block number into a physical track and sector, operate the drive, and deliver the data to the selected buffer location in RAM.

As part of the startup process, ProDOS writes the entry address for each Disk Device Driver into the appropriate positions in its table of Disk Device Driver Vectors. Therefore, once the initialization process is complete, all ProDOS has to do for data transfers is set up locations $0042 to $0047 and jump through the appropriate vector in its Global Page table.

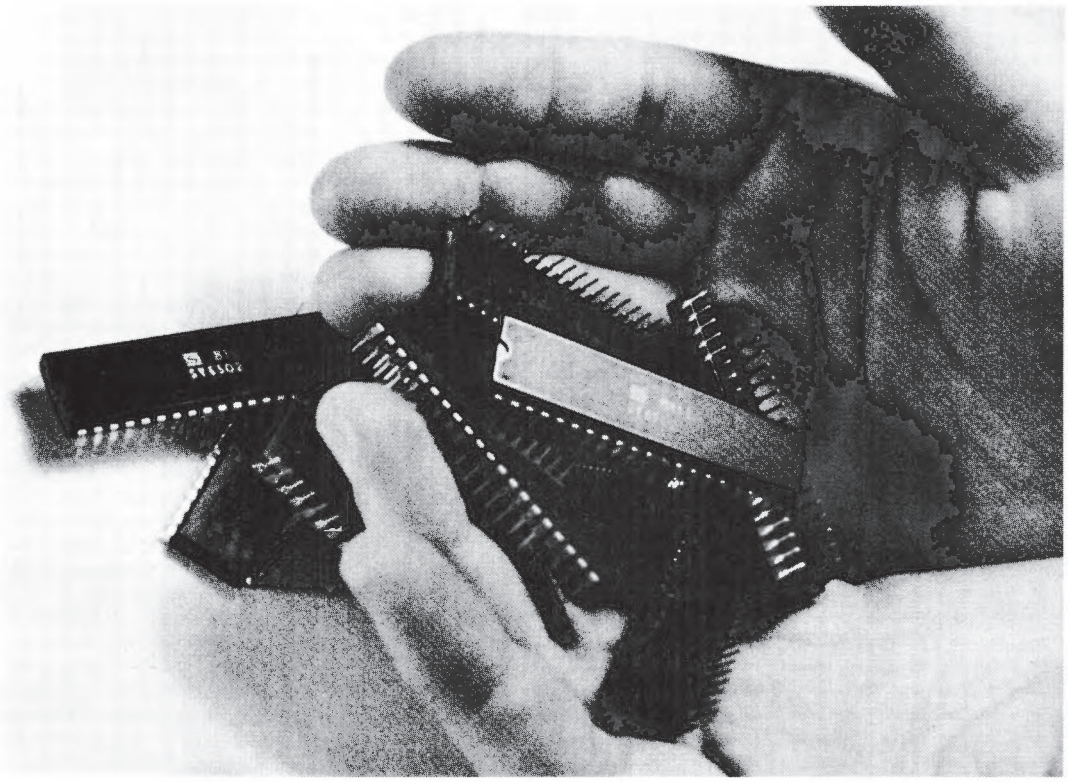
Because the standard Disk II Interface Card does not have such a program on it, there is an expanded and modified version of RWTS included with the ProDOS package, and it is vectored into the device address table automatically if ProDOS detects a Disk II controller card. Owners of other kinds of specialized disk interface cards that were designed before the introduction of ProDOS may not be so lucky, but it should be possible to load in an appropriate patch routine or install a new ROM on such cards.

The beauty of this system is that all you have to do to patch in a powerful new high capacity disk system is plug in the card. ProDOS takes care of the rest automatically. All drives look identical for software running under ProDOS except that some have more blocks than others. If you want to use a fancy external drive with the //c, of course, you can't update ProDOS by plugging in an interface card. However, ProDOS can use its Disk Device Drive Vector system to locate and use a special set of hardware control routines loaded into RAM instead of using ROM on an interface card. This is not as tidy, but it works just as well.

Installable Disk Drivers for MS-DOS 2.0

Since Tim Patterson was working from CP/M as a model, he set up the initial design of the ancestor of MS-DOS to have a structure very similar to the CP/M BIOS. However, since IBM provided a disk BIOS in ROM for its IBM PC, it was always quite a bit more tedious to add disk driver routines to MS-DOS 1.1 than it was to add them to CP/M.

One of the most important changes in MS-DOS 2.0 is the implementation of a fairly straightforward means of adding new disk device drivers to MS-DOS. This system is not nearly as elegant and tidy as the ProDOS system. To install a new driver in ProDOS, you just shove your new controller card into its slot, while in MS-DOS 2.0 you may still need to locate an assembly language programmer. Nonetheless, MS-DOS 2.0 is a substantial improvement over MS-DOS 1.1 in this regard.



Chapter 37

Comparing Operating Systems: User Interface

Commands, Calls and Menus

Each of the four operating systems has a very different feel for both programmers and applications users. The user interface for DOS 3.3 is dominated by the DOS Command Parser which is designed to look very much like the Applesoft BASIC Interpreter from the point of view of the user. As outlined earlier (see Chapter 34), the DOS Command Parser runs in tandem with the Applesoft Interpreter and scans all immediate commands and printed output to spot for DOS commands.

The DOS Command Parser uses the information from the incoming commands to set up a File Manager Parameter List and then calls the File Manager. The File Manager carries out the instruction by repeatedly setting up an RWTS Parameter List and calling RWTS. An assembly language programmer with enough information from such sources as *Beneath Apple DOS* by Worth and Lechner can skip the DOS Command Parser and talk directly with the File Manager or directly with RWTS, but this sort of thing is only for advanced programmers and DOS was never designed to be used this way.

The CCP for CP/M Users

CP/M comes with a minimal user interface for applications users, and a second, very elaborate interface designed only for assembly language programmers to use. Digital Research and the companies that sell Z-80 boards all provide extensive documentation on how an assembly language programmer can operate the Basic Disk Operating System (BDOS) with the assembly language interface. The user interface, for the vast majority of folks who are not assembly language programmers, is based on a very spare command parser called the Console Command Processor (CCP). The CCP prompts the user for the name of a program to run. The various utilities that come with CP/M then issue the machine language calls to the CP/M BDOS.

The collection of utilities which you can call by name from the CCP provides information about the system and asks you questions about what tasks you would like CP/M to perform. Some of these programs are loaded into RAM automatically whenever the CCP is loaded. These include DIR (for a list of files on the disk), ERA (to erase a file), TYPE (which lets you see a text file printed out on the screen), and a few others.

The CCP and this small command set are overwritten and erased whenever a program is loaded. However, all programs end by telling the BDOS to reload the CCP, so whenever you're not inside a program you are looking at the A> prompt of the CCP. The important distinction from the DOS Command Processor is that the services of the CCP are not usually available to a running program and are not available at all from Microsoft BASIC. The CCP serves only to issue machine language BDOS commands when no other program is available to help the user. The Microsoft BASIC Interpreter is designed to issue machine language BDOS calls directly in response to its own set of disk commands that don't necessarily resemble CCP commands. In MBasic, for instance, you issue a FILES command instead of a DIR when you want to see a list of files on the disk.

As with DOS 3.3, a CP/M user is provided with a set of additional programs to load from disk which can be used to copy files (CP/M's PIP is comparable to DOS's FID) or to physically copy an entire disk. Particularly in the Banked Version of CP/M Plus, these supplementary programs can provide many dozens of options not available to DOS users. Further, the Banked system can be used to keep many of these utilities in active RAM for instant call whenever the CCP is active. The Gold Card from Digital Research is the only Z-80 card currently available for the Apple which can support the full Banked CP/M Plus command set.

Calls to the CP/M BDOS

To use the operating system services of CP/M, an assembly language programmer needs to know little or nothing of the internal details of the BDOS and BIOS. In CP/M 2.2, there are just 39 services available, each identified by a function number between 0 and 40 (numbers 38 and 39 are not available in CP/M 2.2). Of these, a call to function 0 is approximately equivalent to hitting Control-C, and functions 1 through 12 do the work of the Apple Monitor's GETLN, COUT and KEYIN routines.

The remaining 26 services provide direct control over such elemental file and disk tasks as opening a file, reading a record from a random text file, and setting a file to read only status. The Microsoft SoftCard Programmers Manual provides a brief explanation of each of these services, but anyone interested in working with the CP/M BDOS should consider getting *Inside CP/M: A Guide for Users and Programmers* by David Cortesi (Holt, Rinehart and Winston) and, for even more detail, *The Programmer's CP/M Handbook* by Andy Johnson-Laird (Osborne/McGraw Hill). CP/M Plus offers an additional 25 functions, and the best source on these is the *CP/M Plus Programmer's Guide* from Digital Research.

The calling convention for BDOS Calls relies on the fairly large number of general purpose microprocessor registers available in the 8080 (see Chapter 29, Figure 29.1). The programmer puts the function number in register C, and any required parameters in register E, or registers D and E if it is a two byte parameter. The most commonly used parameter is simply a pointer to the relevant File Control Block. Once these three registers are set up and the FCB or other list of parameters is ready to go, the program does a jump through location $0005 near the bottom of the zero page, and the BDOS goes into action. If there is any result it must pass back to the calling program, it puts it in register A or, if it is two bytes, in registers H and L.

The MS-DOS Command and Call System

The programmer and user interfaces included with MS-DOS 1.1 are strikingly similar to those available with CP/M. If you are a CP/M user, you can boot up an MS-DOS 1.1 system and begin loading and transferring files without ever noticing you're using a different operating system. This is, in fact, exactly what Tim Patterson had in mind when he designed the interface. There was much more concern with convincing people they didn't have to learn anything new than with correcting some of the limitation of the taciturn old CP/M command system. There are no menus, help facilities, etc.

The function call conventions are also quite similar. The principal difference is that MS-DOS takes advantage of the elaborate interrupt vector system of the 8088 and the INT instruction which can generate software interrupts. The selected call number and the appropriate parameters for the call are placed in specified 8088 registers, and then an INT $21 instruction is executed. This causes an interrupt, and it causes the 8088 to look at the contents of the position number $21 in its table of interrupt vectors. MS-DOS keeps the entry location to its function call handler in vector position $21 and so goes to work on your request. Use of an interrupt allows the convenient resetting of the Code Segment Register so that the operating system does not need to take up space in the same segment where your program is running.

Microsoft Windows

With larger amounts of available memory, concurrency (see Chapter 32) has become important for microcomputers. This means that you can have several programs active at the same time, and switch among them at will. To ease the user through the additional complexity, operating system designers have been turning to the window metaphor in which each of the active programs can be given some of the space on the CRT screen.

Although the visual representation is helpful in keeping track of several tasks, windows can be confusing in that they introduce extraneous information to the screen. The windows in Lisa and MacIntosh take advantage of several subtle visual cues to make the screen seem uncluttered (see Chapter 8), but clutter is a bit of a problem with the Microsoft interface.

In the ideal environment, a single program could handle all of the possible tasks you might want to use the computer for. In that situation, the different tasks would be integrated in two senses: the command structure would be the same for all the tasks, and data from one task could move instantly into another (i.e., inventory, to graphics, to spreadsheet, to word processor, stopping only to give different commands). Unfortunately, you can't get software like that yet although the Lisa software tends strongly in that direction.

The idea of concurrency and windows independent of actual integration is what Microsoft Windows is all about. It lets you move easily from program to program, but does not guarantee any level of similarity or ease of data exchange among the programs. Microsoft hopes that programmers will take advantage of the common framework provided by Windows to work toward higher and higher levels of integration with other programs.

Three Interfaces for ProDOS

The underlying structure of the user interface in ProDOS is closer to CP/M than it is to DOS 3.3, in that the part of ProDOS which remains in memory at all times can only be addressed in machine language. Apple provides several support programs which make the full services of ProDOS available to the non-programmer. These support programs play the same role as the CP/M CCP, but they are all menu driven and so much easier to use than the CCP with its command line structure.

For BASIC programmers, there is a program called BASIC.SYSTEM which acts very much like the DOS Command Parser. It runs in tandem with the Applesoft Interpreter, intercepting both immediate and deferred commands (see Chapter 34). When the BASIC.SYSTEM is loaded and running, ProDOS looks to the user as if it were nearly identical to DOS 3.3. There are, however, a number of differences in commands between DOS and the ProDOS BASIC.SYSTEM (six commands gone, 14 modified, and seven new ones added), so you should get some sort of manual for ProDOS BASIC before switching over from DOS.

In addition to this DOS-like user interface for BASIC programmers, there is also a new Machine Language Interface (MLI) for assembly language programmers which Apple hopes will prove as popular and as efficient as the CP/M BDOS has been for top flight commercial programmers. Finally, there are two menu-driven user interfaces for applications users.

The ProDOS Kernel and the MLI

The main part of ProDOS can be referred to as the ProDOS Kernel to indicate that it is the core of the system and must always be present for ProDOS to operate. In ProDOS 1.0, the Kernel is nearly 15K in length, so it is larger than the DOS 3.3 File Manager, Command Parser and RWTS put together. The entire Kernel is loaded into the high 16K of bank switched RAM (see Chapter 25), and so the system will not work on a 48K Apple.

The two major parts of the Kernel are its Block File Manager (BFM), and its Interrupt Handler (IH). The whole system is supervised by a Command Dispatcher which selectively calls the various routines in the BFM and the IH. Therefore, one operates ProDOS by passing commands to the Command Dispatcher.

Apple has chosen to take a determined stand against releasing any information about the internal routines of the Command Dispatcher, the BFM and IH. This stance may be a little unpopular with some programmers, but most Apple owners should support this position. During the first two years after Apple DOS was released in June of 1978, it was revised, expanded, and improved five times (DOS 3.0, 3.1, 3.2, 3.2.1, and, finally, DOS 3.3). However, commercial programmers began to make such heavy use of internal routines of DOS 3.3, that Apple was forced to stop improving and expanding it for fear of making it incompatible with some important commercial software. In the long run, this has hurt Apple and it has hurt Apple owners.

To attempt to avoid a repetition of these problems Apple has provided a standardized Machine Language Interface (MLI) which it promises will not change. The MLI is simply a set of command codes by which a programmer can address the Command Dispatcher. Although the MLI will remain fixed, Apple has retained the right to freely alter and improve the internal routines of the BFM, the IH, and even the Command Dispatcher itself.

In the future, Apple may add additional MLI commands, but the system will retain compatibility with older software as it grows. Of course, any assembly language programmer can disassemble ProDOS on their own to learn about the internal routines, but the MLI is sufficiently versatile that Apple hopes they won't want to.

Calling Conventions for the MLI

The MLI calls are divided into eight Housekeeping calls which manipulate directory entries, twelve Filing calls which direct the BFM's actions during data transfers, and five System calls which provide direct access to hardware control routines at a fairly low level. Of the five System calls, one is for reading the time, two deal with interrupts, and the remaining two can be used to read or to write a single block from a ProDOS or from a DOS 3.3 diskette.

To use an MLI call, the programmer sets up a table of parameters which usually include a file's name or Reference Number (see Chapter 35), as well as a list of codes for the action ProDOS is supposed to take. Each call has its own requirements for the contents of the parameter table, and these tables can be placed anywhere in free memory.

Once the table of parameters has been set up, the call is actually made by sending control to the following instruction sequence:

```
JSR     MLI         ; where MLI is always $BF00
DB      CMDNUM      ; one byte to select one of the calls
DW      CMDLIST     ; two bytes point to the parameter table
BNE     ERROR       ; branch to error routine if necessary
```

The first line causes a jump to the Command Dispatcher, and the first thing the Command Dispatcher does is to look into the stack to find out where the call came from. When it finds the responsible JSR MLI instruction, it knows that the next byte in memory will be the number of the call it is supposed to carry out, and that the following two bytes will point to the location of the parameter table. It can then look through the parameter table to get all the details, and then set up the BFM or IH to take action. Any results which must be reported back to the calling program will get stored in the caller's parameter table when ProDOS is finished. The BNE instruction is a way for a program to learn that ProDOS did not succeed in carrying out the instruction.

Startup Files for the Menu-Driven Interfaces

When ProDOS is booted, it loads and then runs through an initialization routine, some of which was described in Chapter 36. Once the Kernel is in its final position, and the Global Page vectors and variables have been set up, ProDOS scans the Volume Directory for the first file with a file type of SYS and with .SYSTEM at the end of its name. The file type of SYS indicates that it is a program which can issue MLI calls, the inclusion of .SYSTEM in the name is just a flag to see that it is loaded first.

The BASIC.SYSTEM file accepts commands which closely resemble DOS 3.3 commands and responds by issuing the appropriate MLI calls to the Command Dispatcher. Once it is loaded, a program in BASIC can use ProDOS. If BASIC.SYSTEM is not loaded and running, a BASIC program cannot get any response out of ProDOS.

The BASIC.SYSTEM file has its own initialization process, and when it is finished, the last thing it does is to scan the Volume Directory for a file with the file type of BAS (indicating

that it is a program in BASIC) and the name STARTUP. You can assign the name STARTUP to any BASIC program. It is automatically loaded and run at the end of the full boot process just as a HELLO program was under DOS 3.3.

In this fashion, the full ProDOS boot sequence can be arranged to load the operating system, handle all internal configuration chores, and then load and run a user friendly menu system that lets the application user control ProDOS without knowing any commands.

The menu system originally distributed with ProDOS for the Apple II/II+ and //e comes on a disk called /USERS.DISK. It provides access to a wide range of ProDOS operations, but it is a bit cumbersome to use, particularly for people who are new to computers. When the //c was released, it came with a wonderfully enhanced user interface on a disk called ///c.UTILITIES that is a delight to use. It lacks a few of the operations included in /USERS.DISK, but these are more than made up for by the ease of use.

The /USERS.DISK Main Menu

On the ProDOS/USERS.DISK that is issued by Apple, the STARTUP program operates the ProDOS Main Menu (see Figure 37.1). From the Main Menu, you can choose to exit to the square bracket prompt of the Applesoft Immediate Mode, but you do not do this unless you are planning to do programming in Applesoft BASIC. The great majority of people who rarely program in BASIC are thus spared being forced to enter the Applesoft Interpreter every time the Apple is turned on.

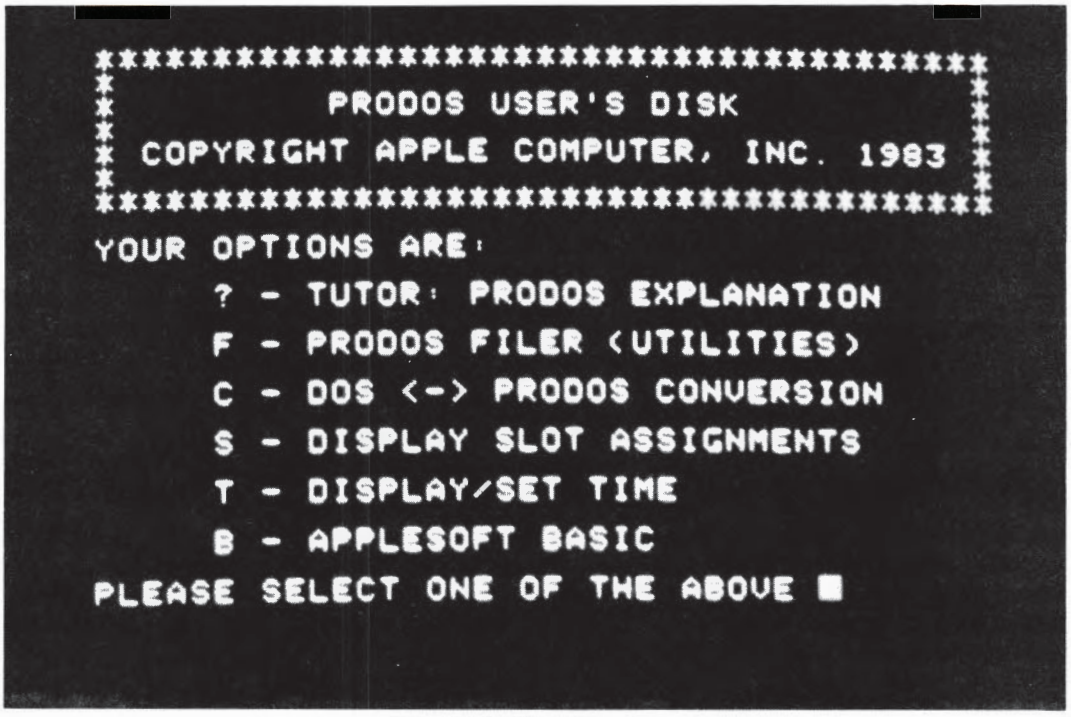


Fig. 37.1 ProDOS main menu.

The Main Menu lets the user get a report of what peripheral cards are installed and also let's the user set the date and time if there is no ThunderClock installed, but its principal use is to gain entry into one of three other interfaces. One is the entry to Applesoft BASIC already mentioned. The other two possibilities lead into additional menus. The "C" option is one that will not be used too frequently. It leads to a brief menu of options for converting DOS 3.3 disks into ProDOS disks and vice versa. The "F" option leads to the ProDOS Filer, and it will probably be the most common route taken out of the Main Menu.

The ProDOS Filer

When the "F" option is selected from the Main Menu, ProDOS locates the Filer system program on the disk. This is not a program in BASIC like the Main Menu; rather, it is a machine language program which completely overwrites and erases BASIC.SYSTEM when it is loaded. The Filer is actually a much larger program than the ProDOS Kernel, and in fact is twice as large as BASIC.SYSTEM.

For the most part, the Volume and File sections of the Filer Menu system provide direct menu driven equivalents for most of the MLI calls. One important difference is that the routines for formatting a Disk II drive can be used only from the Filer and are not available to machine language programs when the Filer is not loaded. Apple will provide commercial programmers with a stripped down version of their ProDOS Disk II formatter, but for the most part, Disk II disks must be formatted from the menu. The lack of an MLI call for formatting also means that there is no longer a command available at the Applesoft Immediate Mode for formatting disks.

You can exit from the Filer only by selecting the Quit option, but this will cause ProDOS to overwrite the Filer with BASIC.SYSTEM, and then STARTUP will be run once again to redisplay the Main Menu. From here, you can exit to the Applesoft Immediate Mode, clear the STARTUP program from memory, and begin to work much as you would with DOS 3.3.

The //c.Utilities Disk

Anyone who has used DOS, CP/M, Pascal, MS-DOS, or even the original ProDOS /Users.Disk is guaranteed love at first sight when given the chance to use the //c.Utilities Disk. With all the hoopla and commotion surrounding the release of the //c, surprisingly little attention has been paid to this wonderful piece of breakthrough programming.

This is without a doubt the most skillfully executed user interface ever conceived for the Apple. Now that it exists, it should serve as a model for all other applications programmers. The /Users.Disk described above came from the company that gave us DOS 3.3 and Pascal 1.1, but the //c.Utilities Disk came from the company that gave us MacIntosh and Lisa.

Firstly, this is a huge program by Apple standards. The initial load stuffs nearly 100K of program and preprinted screens into RAM. It is all done with ProDOS' fast loading technique, so this takes only about 25 seconds. After the load, everything is in RAM, and all features are available instantly. This gives the all important sense of spontaneity to the execution of any choice, and the feeling of raw computer power since so many dozens of things can happen so quickly.

Virtually all options including formatting, copying files, setting up serial ports, scanning directories on DOS, CP/M, Pascal, or ProDOS disks, etc., can be done using only the arrow keys and the Return key. With one hand resting in the lower right corner of the keyboard, you can hit all the arrows, and you can also hit the Solid-Apple and question mark (?) key used for help windows. Yes, windows and also icons.

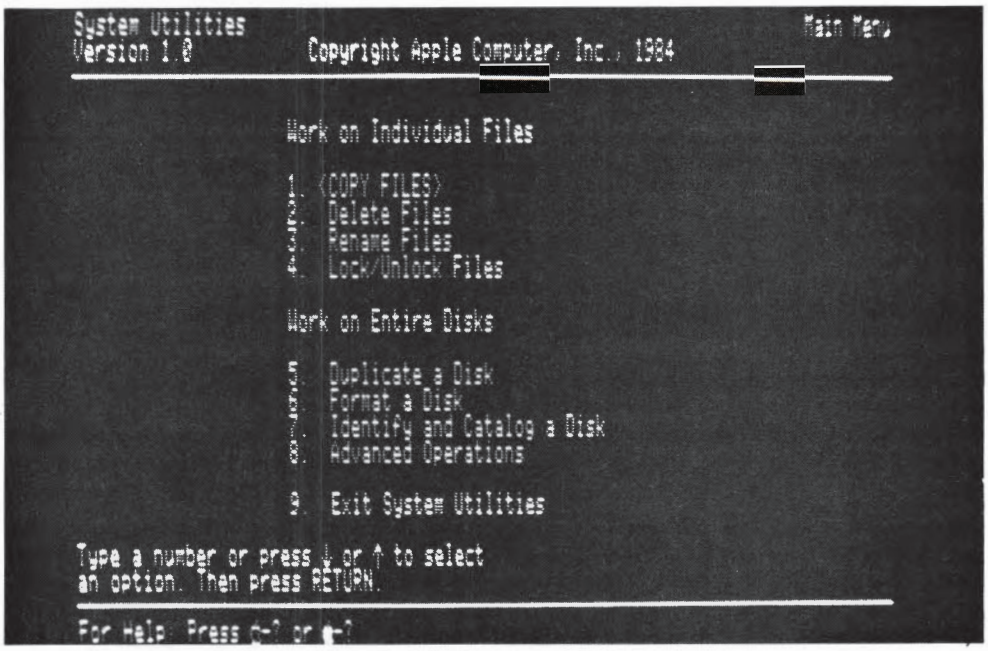


Fig. 37.2a The //c utilities main menu.

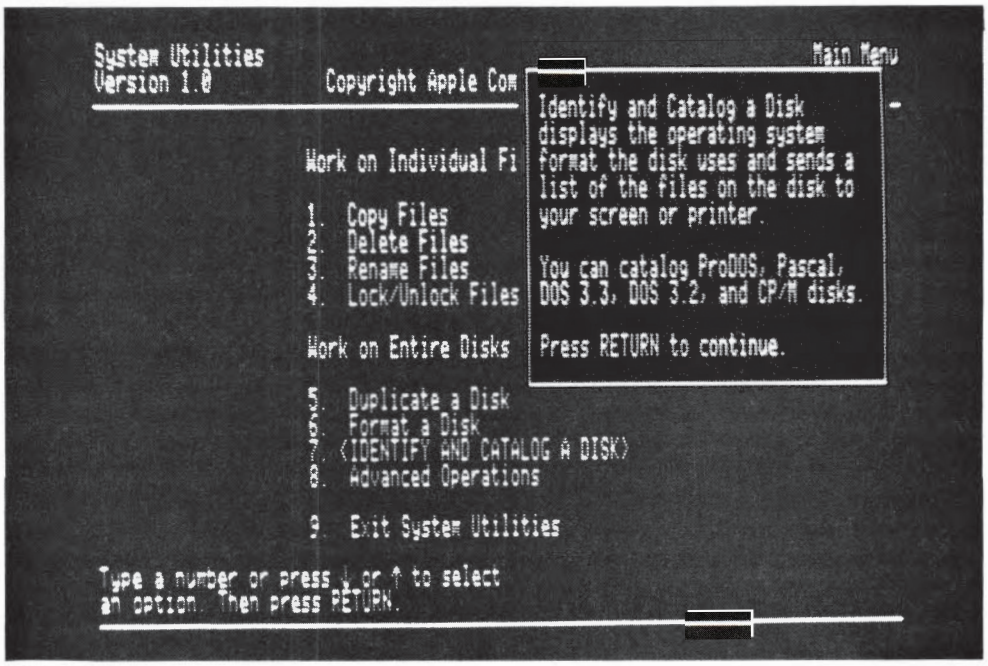


Fig. 37.2b The //c utilities main menu with help window.

The primary menu is shown in Figure 37.2a. As you tap the arrow keys, the brackets and capitalization move to confirm your selection. You can see the Open and Solid Apple icons at the bottom of the screen. These are from the MouseText character set (see Chapter 26, Figure 26.11 and Chapter 26, Table 26.6). When you do request help, a window (see Figure 37.2b) splashes instantly on the screen. There are pop up help windows instantly available for all options. The speed at which they appear and disappear reveals fast block moves of preprinted panels.

When you need to select individual files for copying, deleting, etc., you still can do everything with just the arrow keys. The entire catalog listing of your disk is copied in and displayed on the screen. In Figure 37.3, the arrow keys have been used to select four of the files. Each of the selected files is marked with a check mark from the MouseText icon set.

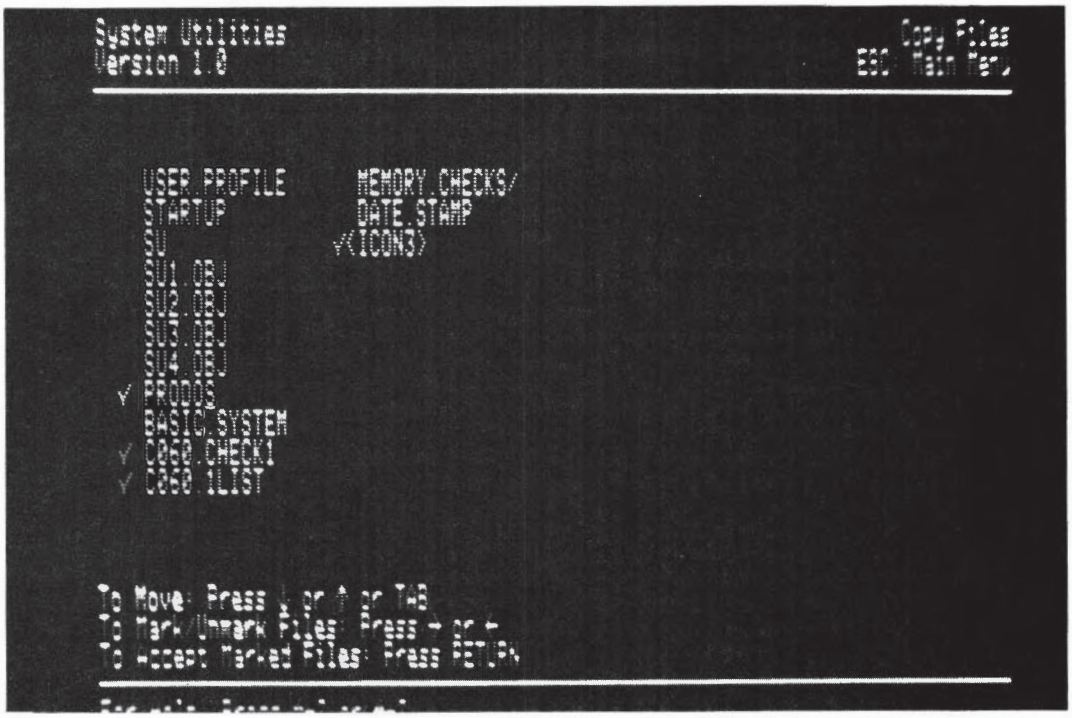


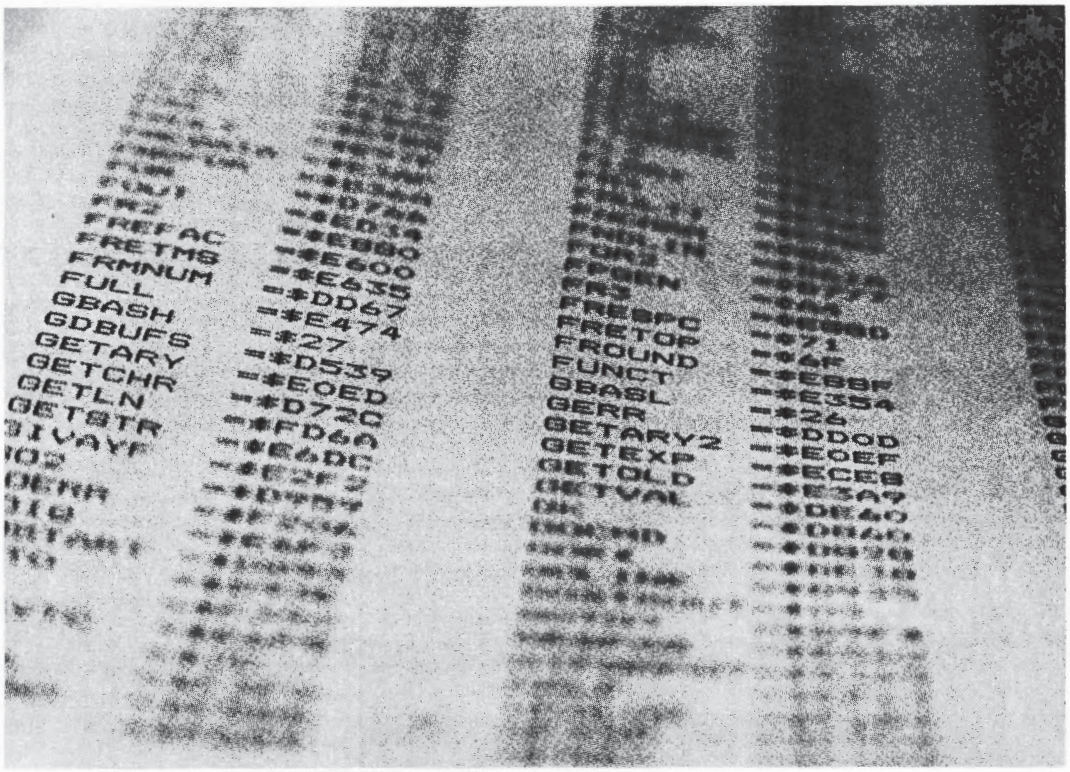
Fig. 37.3 File selection for copying with //c utilities disk.

The only situation in which you must actually type something is when you are naming or renaming a file (or disk). Here again, the user interface people have carried out a wonderfully obvious improvement. The cursor acts like a word processing cursor. You use the arrow keys to position the cursor, but only the Delete key can actually erase anything. If you type a new character over an old one, it is inserted and the rest of the word pushed to the right. When you hit Return, the entire edited word is captured, not just that part to the left of the cursor.

What, you may ask, is so special or difficult about providing a standard word processing cursor on a microcomputer? Good question. See if you can find anything like it in the user interfaces for CP/M, MS-DOS, Pascal, DOS, or even other versions of ProDOS.

PART 4

The Interpreter Apple


FREFAC
FRETMS
FRMNUM
FULL
GBASH
GDBUFS
GETARY
GETCHR
GETLN
GETSTR
=*E600
=*E635
=*DD67
=*E474
=*27
=*D539
=*E0ED
=*D72C
=*FD66
FRESPC
FRETOP
FROUND
FUNCT
GBASL
GERR
GETARY2
GETEXP
GETOLD
GETVAL
=*71
=*4F
=*E354
=*24
=*DDOD
=*E0EF
=*ECE8
=*E3A9

Chapter 38

The Applesoft Interpreter

The Applesoft BASIC Interpreter

A program written in Applesoft BASIC is stored in the Apple's memory in a form which is completely unintelligible to most humans, and which is equally unintelligible to the Apple's 6502. Each line of the program has been labelled, numbered, tokenized, compressed and encoded. Some of the information can be quickly translated into human readable form, other parts are intended for creating pieces of 6502 machine language program. Together, the various elements of each line make complete sense only to a sort of third party within the Apple which talks both to humans and to microprocessors. This third party is the Applesoft BASIC Interpreter.

The Interpreter can translate from its own cryptic code back up into the English-like statements of a BASIC program just as easily as it can translate from its own code on down into 6502 machine language. The benefit of this odd hybrid Interpreter code is that it lets the Interpreter serve as an interactive mediator between the 6502 and a BASIC programmer who knows nothing of the internal makeup of the computer. It guides the human and the 6502 through the program together on a line by line basis. Whenever the 6502 encounters difficulty executing the program, the human can instantaneously see the English equivalent of the exact statement that caused the problem.

The Interpreter can always be directed to translate any numbered line in the coded program. For instance, if it is sent to line 130 with a LIST 130 it will produce the English-like translation of that line and print it out on the display screen. If, on the other hand, it is sent to line 130 with a GOTO 130, it will generate the machine language translation and cause the microprocessor to execute it.

The Structure of Program Line Fields

In Apple memory, the lines of a Basic program are stored in a series of Program Line Fields (PLFs).These Program Line Fields can vary in length from as few as six bytes to a practical maximum of 245 bytes (although this can be extended), and the last byte of each field is set to zero ($00). This zero byte is also referred to as the End of Line (EOL) marker.

All of the Program Line Fields in a program are strung together in a linked list format as shown in Figure 38.2. This means that the first two bytes of each PLF are actually a pointer to the address at which the next PLF begins. These pointers are helpful in some situations in which the Interpreter needs to quickly scan through a series of lines without actually

reading through the entire contents of each program line, and they also provide the basis for assigning an absolute location in the 6502's address space for the beginning of each program line.

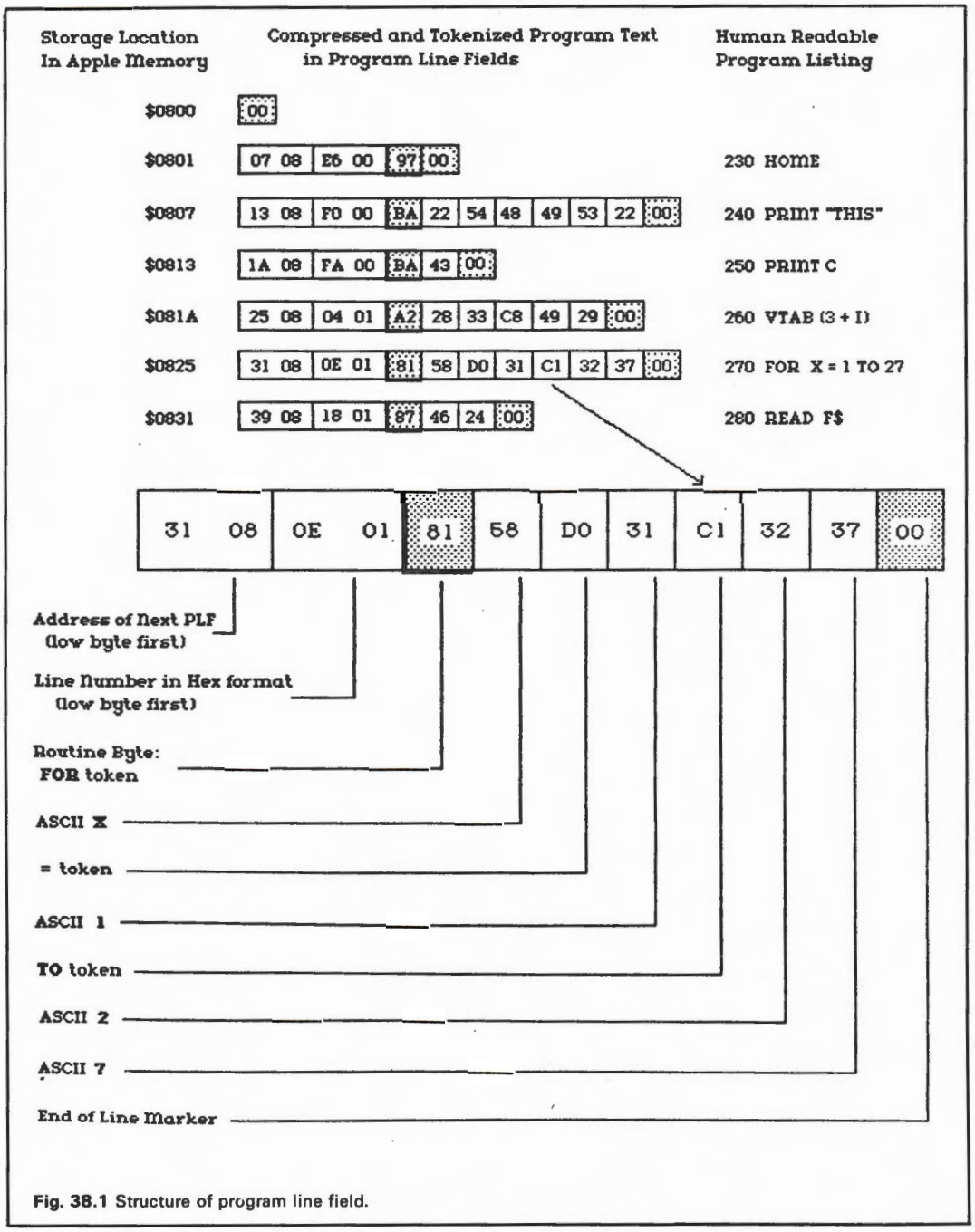


Fig. 38.1 Structure of program line field.

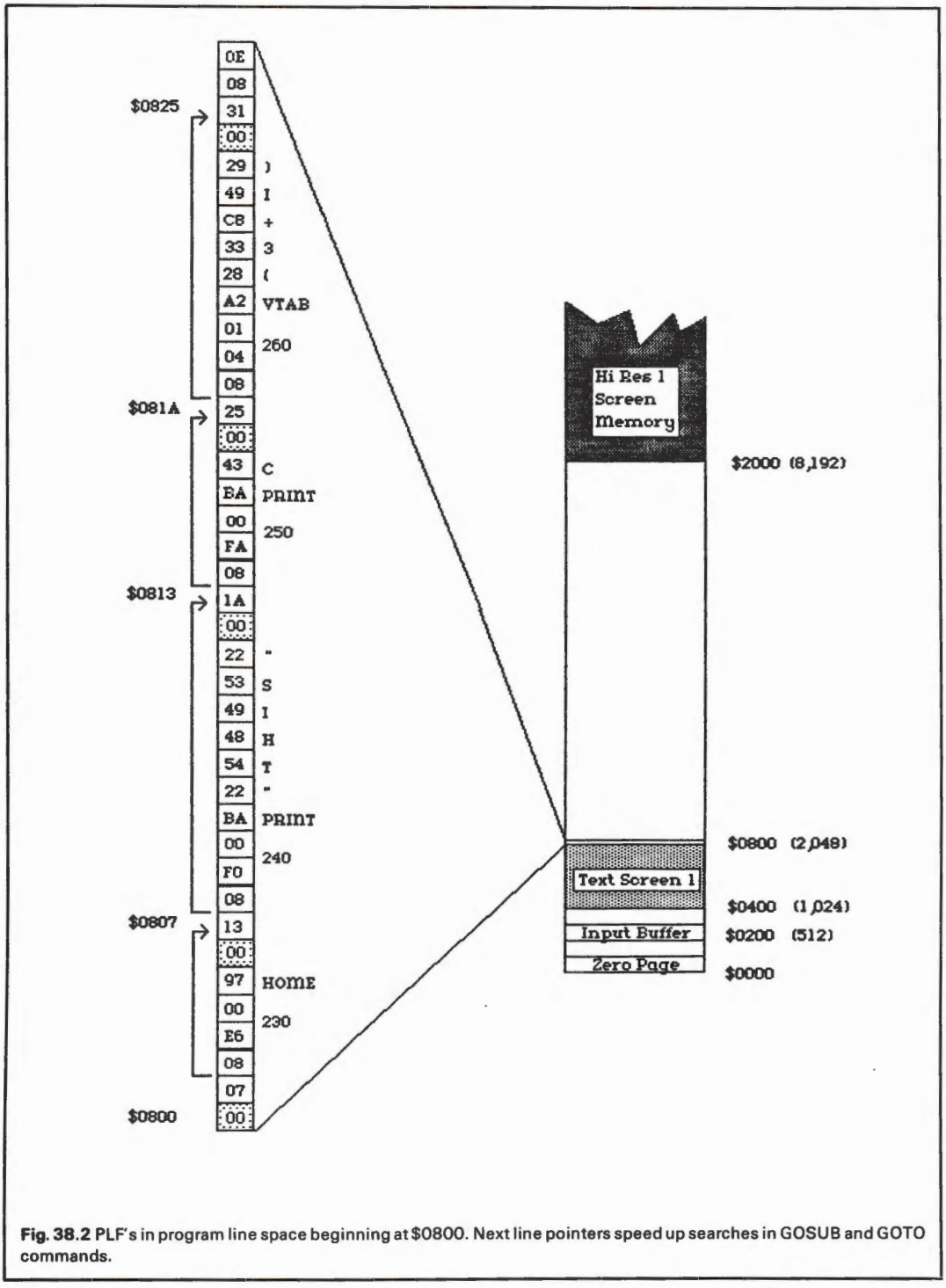


Fig. 38.2 PLF's in program line space beginning at $0800. Next line pointers speed up searches in GOSUB and GOTO commands.

The third and fourth bytes of each field contain the hexadecimal equivalent of the Line Number typed in by the person writing the BASIC program. Note that this is different from the ASCII representation. For instance, the decimal number 24560 is a legitimate line number whose ASCII representation is: $32 $34 $35 $36 $30 (see the ASCII code table in Chapter 18, Table 18.1), but whose numerical representation in hex is: $5FF0. Therefore, any line number between 0 and 65,535 can be represented as a two byte hexadecimal number, and these are the two bytes placed in the third and fourth byte positions of each Program Line Field (see Figure 38.1).

The number of bytes between the Line Number and the EOL marker varies depending on what the programmer put in that line. Each PLF contains at least one Program Statement in that space, and it may contain several statements as long as they are each separated by a colon (ASCII $3A). The first byte of each Program Statement is actually a coded pointer to some internal Applesoft command execution routine, so it may be termed a "Routine Byte."

The Interpreter's View of the Program Line Fields

Although the 6502 sees the program as a linked list of fields following each other one after another in memory (see Figure 38.2), the Applesoft Interpreter sees them a little differently, as shown in Figure 38.1. The representation in Figure 38.1 resembles what the Interpreter has to work with when it is told to LIST 130 or GOTO 130. It starts at the bottom of the pile of PLFs and uses the linked list pointers to step on up through the program.

On arrival at each line, the Interpreter can take a quick look at the Line Number stored there. If it is not the Line Number it is looking for, it reads the pointer there to find out where the next PLF begins, moves to the beginning of the next line, and so on. When it finally finds the correct Line Number, the pointer value it is currently holding is the actual 6502 address of the PLF it is supposed to act on.

The RESTART Loop for Program Entry

The statement space in each Program Line Field contains a compressed and coded version of the line that the BASIC programmer types at the keyboard. The compression and coding is done immediately after you hit Return when you are typing in a new program line, and it is carried out by a set of subroutines in the RESTART ($D43C) loop of the Interpreter. (As explained in Chapter 28, you can use some programs provided with the Merlin Assembler from Southwestern Data Systems to produce an unofficial commented source listing of the Applesoft Interpreter if you want to actually work your own way through the program steps. The Interpreter is essentially identical in the II+, //e and //c.)

Each pass through the loop begins with a call to the GETLN system which is described in detail in Chapter 33. Briefly, GETLN is the Apple's way of collecting a complete line of typed characters from the keyboard. It puts the appropriate prompt (such as the bracket "]") on the screen to let you know what environment you're working in and puts a cursor on the screen to let you know it's OK to start typing.

While you are typing, but before you hit the Return key, GETLN allows you to do a little editing with the arrow keys, and it echos each character you type onto the video screen so you can see what you're doing. The characters also get put into a 256 byte Input Line Buffer

at $0200 to $02FF (decimal 512 to 767). When you finally hit Return, GETLN leaves the characters in the Input Line Buffer and returns control of the 6502 back to the program that called it, in this case, back to the Interpreter's RESTART routine.

Once GETLN is done, the various parts of RESTART begin to scan through what you've left in the Input Line Buffer. They go about rearranging and compressing what you typed and overwriting the Line Buffer's contents with the compressed version. The two things that can be done to compress the line are remove all spaces between characters, and replace all recognizable commands, such as PRINT or VTAB, with a one byte "token" that can be decoded later if necessary.

If your line began with a number, then the Interpreter assumes the line is for deferred execution and goes about inserting it into the correct place among the other lines of your program stored in memory. If your line didn't begin with a number, it assumes you want immediate execution, and it turns over control to the Interpreter's execution section. When all this is done, control returns to the beginning of the RESTART loop, and a prompt and cursor appear to let you know it's OK to type in a new line.

Limited Editing Abilities of GETLN

Shortly after the Apple is first turned on, the 6502 begins executing the Applesoft Interpreter, heads into RESTART, and quickly pops back out into the Monitor's GETLN routine. On return from GETLN into the Interpreter, the new line of typed input is immediately subjected to processing and coding.

This in effect means that no part of the Applesoft Interpreter is devoted to editing and preparing the typed program input. The Interpreter merely borrows a convenient routine from the Monitor to collect the input and then makes no further provision for helping the programmer with editing tasks. The modest backspace and screen pick features of GETLN are all the editing support you get.

You do have the option of preparing the original source code with a word processor, taking full advantage of character insertion, search and replace, etc. but then you lose the advantages of developing the program interactively with the Interpreter. There's no reason why an interpreter can't contain a full set of editing routines, and, in fact, the Microsoft BASIC Interpreter, which interprets MBASIC on the Microsoft Z-80 SoftCard, includes fairly extensive editing support as you work. You can't use MBASIC for many of the tasks Applesoft can handle (see Chapter 4), but fortunately you can add editing abilities to the Applesoft Interpreter.

Using GPLE to Add Editing Capabilities to Applesoft

There is a long lineage of add on editors for the Applesoft Interpreter, of which the one in most widespread current use is called the Global Program Line Editor (GPLE). This program works by periodically breaking into the flow of program control while you are typing. It can both edit the current input, and also scan through and retrieve the encoded lines in memory to make them available for editing.

Those of you who have worked through the section on GETLN will know that for the actual keyboard input and for the actual screen output, control leaves GETLN temporarily. To get input, GETLN issues a jump through a location called KSW and this location can be set up to point to any routine the user chooses. Similarly, when GETLN wants to echo a character to the screen, it jumps through a changeable location called CSW (see Chapter 33).

If you read through the discussion of the DOS 3.3 Command Parser (see Chapter 34), you will also recall that DOS intercepts each character that GETLN is trying to send to the screen. When GETLN jumps through CSW, DOS or ProDOS is waiting on the other side to temporarily sieze control of the Apple, examine the character to see if it heralds anything interesting like a DOS command, and if so take action. When DOS is through with its intervention, it can send the character back along its way to the screen and have control return to GETLN.

GPLE works in the same way as DOS, by intercepting jumps through CSW. Therefore, you can type escape sequences and control characters to get GPLE's attention and it will respond by treating the next few incoming characters as editing commands, just as if you were using a word processor. You use GPLE to tidy things up in the Input Line Buffer, and then send the edited output on to the Applesoft Interpreter for standard processing.

You can get GPLE on disk from Beagle Brothers, but in this form it does take up space in the Apple's memory. Hollywood Hardware now offers a card you can install in one of the accessory slots which has GPLE stored in ROM. While it is running, the editing program resides in the "Expansion ROM" space (see Chapter 22) so it does not interfere with the memory requirements of any other program. The ROM based version of GPLE offers cursor moves to the beginning and end of the line, character insertion and deletion, and even a search function within the line. In addition, it offers global search and replace of any string in program memory and macro capabilities. The macro feature lets you enter frequently used strings with a simple two stroke escape sequence. When GPLE sees a sequence it recognizes, it fiddles with KSW and pumps the full string into GETLN as if it were coming from the keyboard.

Clearing the Eighth Bit for ASCII Codes

Once GETLN (and GPLE) are finished collecting a line of text that the programmer terminates with a Return, control bounces back up to the INLIN2 ($D52E) section of RESTART, and processing begins. The first processing routine is GDBUFS ($D539) and it begins the coding process by doing two things. First, it places a value of $00 at the end of the line. This EOL marker will stay with the line for the rest of its life.

The second task is to work its way back through the line and set the eighth bit of every character to zero. A quick check of the ASCII code table (see Chapter 18, Table 18.1) reveals that in standard ASCII, all of the 128 character codes can be represented by using only the low seven bits of a byte. The eighth bit is used by a variety of systems as a marker bit to add a little internal information to each character. All characters coming in from the Apple keyboard have the eighth bit set to one (see Chapter 33), and this must be maintained when the character is sent to the screen or else it will be displayed as flashing or inverse. However, in the Applesoft Interpreter's Program Line Fields, the eighth bit has a very different meaning.

With the exception of this first pass of the GDBUFS routine, the Applesoft Interpreter will not recognize a byte as an ASCII code unless its eighth bit is set to zero. This is because the Interpreter has its own set of 128 Interpreter codes called Tokens which have nothing to do with ASCII and in which the eighth bit is set to one. Within a Program Line Field, any byte with its high bit clear ($00 to $7F, decimal 0 to 127) will be recognized as an ASCII code, while any byte with its high bit set ($80 to $FF, decimal 128 to 255) will be recognized as an Interpreter Token code. Thus, when GDBUFS finishes its work and control returns to INLIN2, the line in the buffer contains ASCII codes with the high bit clear and ends with a null byte ($00) as an EOL marker.

The TXTPTR Pointer and the CHRGET System

Whenever the Interpreter wants to examine a byte from within a BASIC program, it uses a routine called CHRGET to fetch the byte. The CHRGET routine is stored in the Applesoft ROM at $F10B, but it is always copied down into the zero page for use ($00B1 to $00C8, decimal 177 to 200). Within CHRGET, the two bytes at $B8 and $B9 are called the TXTPTR (text pointer), and they provide the address for an LDA instruction at $B7.

To get a program byte for inspection, any routine in the Interpreter can set TXTPTR to point to its location in memory and then do a JSR to the entry point of CHRGET. The CHRGET routine will automatically increment the TXTPTR every time it is called. Therefore, repeated calls to CHRGET will automatically step through a line, scanning each character in sequence. There is an alternate entry called CHRGOT which lets the caller skip the incrementing and just pick up what TXTPTR is actually pointed at.

The CHRGET routine also does a little bit of intial analysis on its own. First of all, whenever it encounters the ASCII token for a space, it increments the TXTPTR and gets the next character without returning to the caller. Therefore, as long as the Interpreter is using CHRGET to scan a line, it will never see a space. It is for this reason that:] 3 4 5 PRINT looks just the same as:]345 PRINT from the point of view of the Interpreter. Obviously, CHRGET wouldn't be too useful for scanning the contents of a literal string (such as "THIS IS A STRING" where spaces must be preserved).

One last service performed by CHRGET is to see if the ASCII code it is looking at is a number as opposed to a letter. This is the basis of the crucial distinction between deferred and immediate execution commands. Whenever the Interpreter starts to examine a new line of input, the first call to CHRGET is used to decide the course of subsequent processing. When the first character is a number, CHRGET returns with the 6502 carry flag cleared and the Interpreter uses this flag to decide what to do next.

If the flag is set, indicating that the first character is a letter, then the line is immediately examined by GETIN ($D559) to see if it is a permissible BASIC command, and then a call for execution of the command is sent through the 6502. If, on the other hand, the flag comes back clear, then the Interpreter begins processing the contents of the Input Line Buffer as a new program line which must eventually end up in its own Program Line Field in the appropriate place in memory.

Initially, the only difference in processing is that the Line Number itself is analysed before GETIN is called to look for commands. To do this, the NXLIN routine at $D464 makes a series of calls to CHRGET to collect up the ASCII codes for the decimal digits of the Line Number as typed in. These are converted from ASCII codes to hexadecimal numbers and summed up appropriately to produce a valid hexadecimal equivalent of the decimal line number (i.e., you see 3841, the line buffer holds the ASCII codes $33 $38 $34 $31, and the routine produces the hexadecimal number $0F01). This is placed in the Interpreter's LINNUM location ($50 and $51, decimal 80 and 81) for future reference, and then GETIN is called to examine the rest of the line for commands.

The Command Tables at the Heart of Applesoft

Within the Applesoft ROM there are two matched tables which are the key to the Interpretation process. One table, called TOKTABL, spells out all the commands just as they must be typed at the keyboard, while the other table, called CMDTABL, is a list of the entry points of the machine language subroutines which can carry out the commands. GETIN has the task of matching the words you type in with the appropriate subroutine entry points in its CMDTABL.

The matching is based on storing, for instance, the word PRINT as the 186th item in the TOKTABL and storing the entry address for the printing subroutine as the 186th item in the CMDTABL. Within the Program Line Field, the PRINT command is represented only by the number 186. This number gives the Interpreter a quick route to the written form of the command in one table as well as a quick route to the address of the appropriate subroutine in the other table.

This command table arrangement is related to one of the most characteristic features of the BASIC computer language in general and Applesoft in particular. A given dialect of BASIC has a fixed set of commands which does not grow as you work, and each of which must be typed with exact spelling to be recognized. One very powerful and popular newer computer language called LOGO does let you add new commands as you work (see Chapter 4), and another language system called Savvy also lets you add new commands but in addition isn't even too particular about how you spell the commands (see Chapter 29). Both of these languages are based on a much more elaborate system of matching words typed at the keyboard to the machine language routines which are available in the language to carry them out.

The Token Table

The TOKTABL, which begins at $D0D0 (53,456), is a long literal list of the ASCII representations of all the commands in Applesoft. The commands are all listed one after another without spaces between them, but they are marked off from each other by the setting of the eighth bit in the last letter of each command. This use of the eighth bit to demarcate commands in the table is just another one of many specialized and unrelated uses of the eighth bit, and, in this case, the bit is cleared whenever one of these characters is copied out of the table to be used.

Although there are no numbers in the table itself, the Interpreter views the table as having 107 entries in a series of numbered positions. The first is position $80 (128), the second is $81 (129) on up to position $EA (234). These position numbers are called Tokens, and, as noted, all command words and symbols are replaced by Tokens within the Program Line Fields. Each Token serves as a sort of pointer to one position in the TOKTABL, and therefore can be decoded by the LIST routine to regenerate the written word for display on the screen.

You can get some idea of what the table looks like in the ROM by copying part of the table into screen display memory as follows. //e and //c owners should first hit ESC CTRL-Q to switch into 40 column mode with the standard character set. Enter the Monitor by responding to the square bracket prompt with: CALL -151 and hitting return. Now type: 0400<D0D0.D11FM which is a Monitor move command to let you see the first 80 bytes of the TOKTABLE. Notice that the last letter of each command is displayed in normal type (high bit set) while the remainder of the letters are flashing (high bit clear; see Chapter 26, Table 26.6).

APPLESOFT TOKEN AND COMMAND TABLE

Token		Keyword	Address	
Hex	Dec		Hex	Dec
$80	128	END	$D870	55408
$81	129	FOR	$D766	55142
$82	130	NEXT	$DCF9	56569
$83	131	DATA	$D995	55701
$84	132	INPUT	$DBB2	56242
$85	133	DEL	$F331	62257
$86	134	DIM	$DFD9	57305
$87	135	READ	$DBE2	56290
$88	136	GR	$F390	62352
$89	137	TEXT	$F399	62361
$8A	138	PR#	$F1E5	61925
$8B	139	IN#	$F1DE	61918
$8C	140	CALL	$F1D5	61909
$8D	141	PLOT	$F225	61989
$8E	142	HLIN	$F232	62002
$8F	143	VLIN	$F241	62017
$90	144	HGR2	$F3D8	62424
$91	145	HGR	$F3E2	62434
$92	146	HCOLOR=	$F6E9	63209
$93	147	HPLOT	$F6FE	63230
$94	148	DRAW	$F769	63337
$95	149	XDRAW	$F76F	63343
$96	150	HTAB	$F7E7	63463
$97	151	HOME	$FC58	64600
$98	152	ROT=	$F721	63265
$99	153	SCALE=	$F727	63271
$9A	154	SHLOAD	$F775	63349
$9B	155	TRACE	$F26D	62061
$9C	156	NOTRACE	$F26F	62063
$9D	157	NORMAL	$F273	62067
$9E	158	INVERSE	$F277	62071
$9F	159	FLASH	$F280	62080
$A0	160	COLOR=	$F24F	62031
$A1	161	POP	$D96B	55659
$A2	162	VTAB	$F256	62038
$A3	163	HIMEM:	$F286	62086
$A4	164	LOMEM:	$F2A6	62118
$A5	165	ONERR	$F2CB	62155
$A6	166	RESUME	$F318	62232
$A7	167	RECALL	$F3BC	62396
$A8	168	STORE	$F39F	62367

Table 38.1 Token and command table.

Token		Keyword	Address	
Hex	Dec		Hex	Dec
$A9	169	SPEED=	$F262	62050
$AA	170	LET	$DA46	55878
$AB	171	GOTO	$D93E	55614
$AC	172	RUN	$D912	55570
$AD	173	IF	$D9C9	55753
$AE	174	RESTORE	$D849	55369
$AF	175	&	$03F5	1013
$B0	176	GOSUB	$D921	55585
$B1	177	RETURN	$D96B	55659
$B2	178	REM	$D9DC	55772
$B3	179	STOP	$D86E	55406
$B4	180	ON	$D9EC	55788
$B5	181	WAIT	$E784	59268
$B6	182	LOAD	$D8C9	55497
$B7	183	SAVE	$D8B0	55472
$B8	184	DEF	$E313	58131
$B9	185	POKE	$E77B	59259
$BA	186	PRINT	$DAD5	56021
$BB	187	CONT	$D896	55446
$BC	188	LIST	$D6A5	54949
$BD	189	CLEAR	$D66A	54890
$BE	190	GET	$DBA0	56224
$BF	191	NEW	$D649	54857
$C0	192	TAB(		
$C1	193	TO		
$C2	194	FN		
$C3	195	SPC(		
$C4	196	THEN		
$C5	197	AT		
$C6	198	NOT		
$C7	199	STEP		
$C8	200	+		
$C9	201	-		
$CA	202	*		
$CB	203	/		
$CC	204	^		
$CD	205	AND		
$CE	206	OR		
$CF	207	>		

Table 38.1 (continued).

Token		Keyword	Address	
Hex	Dec		Hex	Dec
$D0	208	=		
$D1	209	<		
$D2	210	SGN	$EB90	60304
$D3	211	INT	$EC23	60451
$D4	212	ABS	$EBAF	60335
$D5	213	USR	$000A	10
$D6	214	FRE	$E2DE	58078
$D7	215	SCRN(	$D412	54290
$D8	216	PDL	$DFCD	57293
$D9	217	POS	$E2FF	58111
$DA	218	SQR	$EE8D	61069
$DB	219	RND	$EFAE	61358
$DC	220	LOG	$E941	59713
$DD	221	EXP	$EF09	61193
$DE	222	COS	$EFEA	61418
$DF	223	SIN	$EFF1	61425
$E0	224	TAN	$F03A	61498
$E1	225	ATN	$F09E	61598
$E2	226	PEEK	$E764	59236
$E3	227	LEN	$E6D6	59094
$E4	228	STR$	$E3C5	58309
$E5	229	VAL	$E707	59143
$E6	230	ASC	$E6E5	59109
$E7	231	CHR$	$E646	58950
$E8	232	LEFT$	$E65A	58970
$E9	233	RIGHT$	$E686	59014
$EA	234	MID$	$E691	59025
Error Messages Appended to Table				
$EB	235	NEXT WITHOUT FOR		
$EC	236	SYNTAX		
$ED	237	RETURN WITHOUT GOSUB		
$EF	238	OUT OF DATA		
$F0	240	OVERFLOW		
$F1	241	OVERFLOW		
$F2	242	UNDEF'D STATEMENT		
$F3	243	BAD SUBSCRIPT		
$F4	244	REDIMM'D ARRAY		
$F5	245	DIVISION BY ZERO		
$F6	246	ILLEGAL DIRECT		
$F7	247	TYPE MISMATCH		
$F8	248	STRING TOO LONG		
$F9	249	FORMULA TOO COMPLEX		
$FA	250	CAN'T CONTINUE		
$FB	251	UNDEF'D FUNCTION		

Table 38.1 (continued).

Tokenizing the Program Line

When the GETIN routine scans the Input Buffer, it uses the commands stored in the TOKTABL as a series of templates against which to compare what you have typed. It starts with the first command in the table, which happens to be END and checks to see if the first letter you typed is an E. If so, it checks to see if your next letter is an N, and so on. As it steps through the table during the search, it keeps track of the Token number of the command it is currently checking.

Once GETIN has matched up your typed command with one of the commands in the TOKTABL, it can replace the full ASCII representation of the command as written in the Input Line Buffer with the appropriate one byte Token. The use of Tokens saves space since any command can be represented with the one byte code, and it also speeds things up when the program is RUN since the Interpreter can use the Token as a direct index to a routine address in the CMDTABL without having to do any scanning or comparing.

Notice also that all of the Tokens are hexadecimal numbers between $80 and $FF (128 to 255). In contrast, any character in the Input Line Buffer which is not part of a recognizable Applesoft command (i.e., numbers, variable names, DOS Commands, and literals) is represented by an ASCII code with its high bit clear, i.e., a hexadecimal number between $00 and $7F (0 to 127).

This is very handy for the Interpreter when it is trying to either LIST or to RUN a program because it can discover immediately if it should process the byte as a command Token or as an ASCII character simply by looking at the eighth bit. However, this system also has the disadvantage of limiting the language to no more than 128 tokenized commands and symbols.

Inserting a New Line

Once GDBUFS, NXLIN and GETIN have finished their work, the line you typed into the Input Line Buffer has been completely coded and compressed. It begins with a hexadecimal Line Number, it has command Tokens with their high bit set, ASCII character codes with their high bit clear, no spaces, and $00 at the end. It is now ready for insertion into the proper place within the program in memory.

The RESTART loop now calls the FNDLN routine ($D61A) to see if the Line Number you typed in is already in the program. If it finds a line with that number, it erases the old line and then it begins recopying the entire remainder of the program beyond that point, moving it down in memory to fill in the space made by erasure. Only after the move down is complete does it bother to check what you want to put in.

If there isn't a line with the same number, it finds the starting address in memory of the Program Line Field with the next higher Line Number. The NEWLN routine then notes how much space your new line will take up, and goes about making room for it by going to the very top end of the entire program and starting to move everything up in memory. Once everything is moved back up, it finally copies your new line out of the Input Line Buffer up into the space made for it.

Recall, however, that each Program Line Field has a pointer to the next line (see Figures 38.1 and 38.2) and, after all that moving up and down in memory, all the lines above the insertion have got their pointers wrong. The Interpreter must now go back up through the program,

line by line, and reconnect the linked list of pointers. Once everything is linked back up again, control returns to the top of the RESTART loop, GETLN is called, a prompt and cursor appear, and the Interpreter is ready for a new line. With a long program in memory, you can just barely detect the time it takes for all of these events in the RESTART loop to transpire.

The Program Execution Loop

Any program line which has been fully prepared by GETIN can be scanned and executed by the Interpreter. In the case of immediate commands, they are scanned in place in the Input Line Buffer. To carry out a deferred program line, the TXTPTR must first be pointed at the appropriate position in some Program Line Field , at which time the scan for execution can begin. The way you set up the the TXTPTR is to type GOTO followed by the line number you want to start at, or by typing RUN in which case the Interpreter does a GOTO to the first line of the program.

As outlined earlier, each statement in deferred or immediate mode begins with a "Routine Byte." It is this single first byte which is always examined by the main program execution loop. This loop begins with a routine called TRACE? ($D805, decimal 55,301) which can set up the Interpreter to operate in the trace mode for debugging (see the Applesoft Reference Manual), but which is usually passed through without effect.

Using the Routine Byte to Select an Execution Option

The real execution system begins at TR1 ($D81D) with a call to CHRGET to fetch the Routine Byte for the statement to be executed and then a pass through the GOCMD routine ($D828). There are basically four different courses of action that can result. If that first byte is an ASCII character with its high bit clear, then GOCMD assumes that the statement is defining a variable and it jumps to the LET routine ($DA46) which is the front end for all variable handling. This has exactly the same effect as the use of the actual LET command in your program, and this is why the use of LET is optional in Applesoft.

If the Routine Byte is a Token with its high bit set, GOCMD must first check to see if it is an acceptable token. Only the first 64 tokens ($80 to $BF) can appear in the Routine Byte. These Routine Tokens which stand for such commands as PRINT or HPLOT can be used to supervise the remainder of the decode and execution task for any line. The remaining Tokens, which stand for such commands as SQR and PEEK, can only appear within a line, not at its beginning. If one of these other tokens appears in the Routine Byte, then the Interpreter issues a Syntax Error or an Error Message depending on what it was doing when it first noticed the problem.

The remaining two courses of action both involve temporarily turning over control to a command routine. GOCMD uses the Token to select a command routine entry point from the CMDTABL (see above) and then passes control to that routine. The command routine that receives control at this point has full responsibility for reading, decoding and executing the remainder of the statement and then returning to the execution loop.

If the Token is $AF, which stands for an ampersand (&) command, then control passes out of Applesoft into some special machine language routine loaded in by the user. These ampersand routines can provided wonderful enhancements to the command set of Applesoft since the single & Token can lead to the selection of any one of a large number of external routines.

For each of the remaining 63 legitimate Routine Tokens, the CMDTABL provides an address for an effective JSR to one of the main execution routines stored in the Applesoft ROM (see Table 1). Some of the routines are very simple, and do no more than toggling one or two of the Apple's softswitches (see Chapter 26) to change the nature of the screen display or redirect the I/O hooks (PR# and IN#, see Chapter 33). Another group acts on the Interpreter itself, such as GOTO, GOSUB and END, which actually feed back to alter the behavior of the main execution loop, or such as DEL and NEW which act on the RESTART system.

Formula Evaluation and Math

About 20 of the main Applesoft commands are set up so that they can act on some number or string of characters stored as a variable. The variable, number or string they act on can be stated in simple form or can require that a little math get carried out in order to come up with the actual number or string to be acted on. All of the command routines which can act on variables (such as PRINT, HPLOT, VTAB, IF, etc.) begin their work with a call to the Applesoft FRMEVL loop ($DD78, decimal 56,696). Here, all formulas are evaluated to produce a final result which FRMEVL can hand back to whatever routine called it.

The FRMEVL loop works with strings of ASCII characters or with numbers represented in a special kind of numerical format called Binary Scientific Notation (see Chapter 39, Figure 39.2). At the beginning of the FRMEVL loop, calls are made to CHRGET to read in the bytes of the statement to be analysed. Any numbers in the statement are collected and converted from their ASCII character representation into numbers in Binary Scientific Notation. If FRMEVL encounters a variable name, it uses the PTRGET or GETARYPNT routines to fetch the number or string currently assigned to that variable (see Chapter 39).

With the various numbers or strings in hands, FRMEVL then scouts for operators such as the plus sign (+) or asterisk (*), etc., and for more complex operators such as MID$ or COS. Applesoft also provides for an exit from the Applesoft ROM at this point to use a machine language math routine provided by the user. This is signified in your program with the operator USR (X). You get all the services of the FRMEVL variable search and scientific notation subroutines but can carry out your own math operation. When FRMEVL encounters the Token for USR it does a JSR to $000A (decimal 10) where it must find a JMP to the added routine.

The string manipulation abilities of Applesoft are fairly extensive, but they require nothing really remarkable in terms of execution routines. Mathematical analysis, however, is a much more elaborate task, and a large part of the Applesoft ROM is taken up with math routines. Number crunching is one of the traditionally mainstays of computers in general, and languages such as FORTRAN and BASIC had their origins in tasks where calculation was all important. Microcomputers spend a great deal of their time involved in character manipulation and simple math, but Applesoft nonetheless provides some fairly sophisticated calculational powers. In addition, several companies make add on cards which can enhance the mathematical powers of Applesoft (see Chapter 41).

Forty Bit Registers for Floating Point Math

Although all of the calculations performed by Applesoft are actually carried out by the simple eight bit ALU of the 6502 (see Chapters 27 and 28), Applesoft BASIC simulates a much more powerful microprocessor with five math registers each forty bits wide. This fiction is actually just a useful way of organizing the Applesoft math routines and of making it simpler for

assembly language programmers to make use of them in other kinds of programs. It is important because it permits the use of very large numbers, although it is a major speed bottleneck in programs which require a great deal of math.

The Applesoft floating point math registers are scattered around the zero page between $008A and $00AA (138 to 170). The two principal registers are the Floating-Point Accumulator (FAC; $9D to $A3) and the Argument Register (ARG; $A5 to $AA) and there are also three supplementary registers (TEMP1, TEMP2, and TEMP3) for intermediate results. In the FAC and ARG registers, numbers are represented in an expanded binary floating point format as follows: one byte for an exponent, four bytes for a mantissa, and a sixth byte in which the high bit notes the sign of the number.

Pausing for an instant review of scientific notation, any number can be represented as a string of significant digits multiplied by a second number to assign a magnitude. For instance, 3,475 can also be represented as .3475 times 10^4 . For small numbers this just seems to be a nuissance, but for very large numbers it is an utter necessity. You would never want to write out 5.3×10^{38} in its full expanded form; it would just cover your page with unnecessary zeros.

In practical terms the Applesoft number format has the following implications. A four byte mantissa can represent any number from 0 to 16,777,215, so it is equivalent to about eight decimal digits of precision. The one byte exponent can represent numbers from 2^{128} to 2^{-128}, which is equivalent to a decimal number between 3.4 times 10^{-38} and 3.4 times 10^{38} . For a complete explanation of the number format used by Applesoft, you should see *Real Variable Study* by Eric Goetz in *Call - A.P.P.L.E. In Depth: All About Applesoft* from Call - A.P.P.L.E. For assembly language programmers who might want to make use of the floating point math system, there is also a handy review of entry points in *Applesoft Internal Entry Points* by John Crossley in the same publication.

Graphics Plotting Routines

Once FRMEVL is finished analyzing the variables, numbers and strings in a statement, control returns to the command routine which called it. The various routines which call FRMEVL may do so purely to carry out calculations (i.e., $AC = ((EF * RT)/3.45)^3$) or they may just use FRMEVL to prepare the statement preliminary to the main task, i.e., VTAB (X + 4) to calculate a screen position for text output before actually tampering with the cursor control system. In this latter category you would also include the various high resolution plotting commands.

To do plotting on the high resolution graphics screen, Applesoft uses a separate set of plotting routines independent of the FAC/ARG system. The command HPLOT $((XT + 3)^2), ((YT + 5)^2)$ TO 36,125 would first use the FRMEVL loop to evaluate the actual X and Y coordinate to use, and then it would call several high res plotting routines to actually do the plotting. These plotting routines are used to calculate the addresses in display memory where dots should be placed to do the plotting and the proper values to store in those addresses to get the shape just right.

Machine language programs can often draw images much more quickly than Applesoft programs can, but this is not because the plotting routines themselves are faster. Most machine language plotting routines gain speed by skipping straight on past the bottlenecks of the interpretation process and the FRMEVL loop, but end up by simply calling the same built-in

plotting routines that are called by Applesoft programs. If the plotting routines themselves are the real bottleneck in your program, you can't improve on things by switching to machine language or by compiling your Applesoft program (see Chapter 41). If you buy a coprocessor card which only speeds up the FAC/ARG system (see Chapter 41), you may get little or no improvement if your real problem is in the plotting routines.

Ampersand Commands

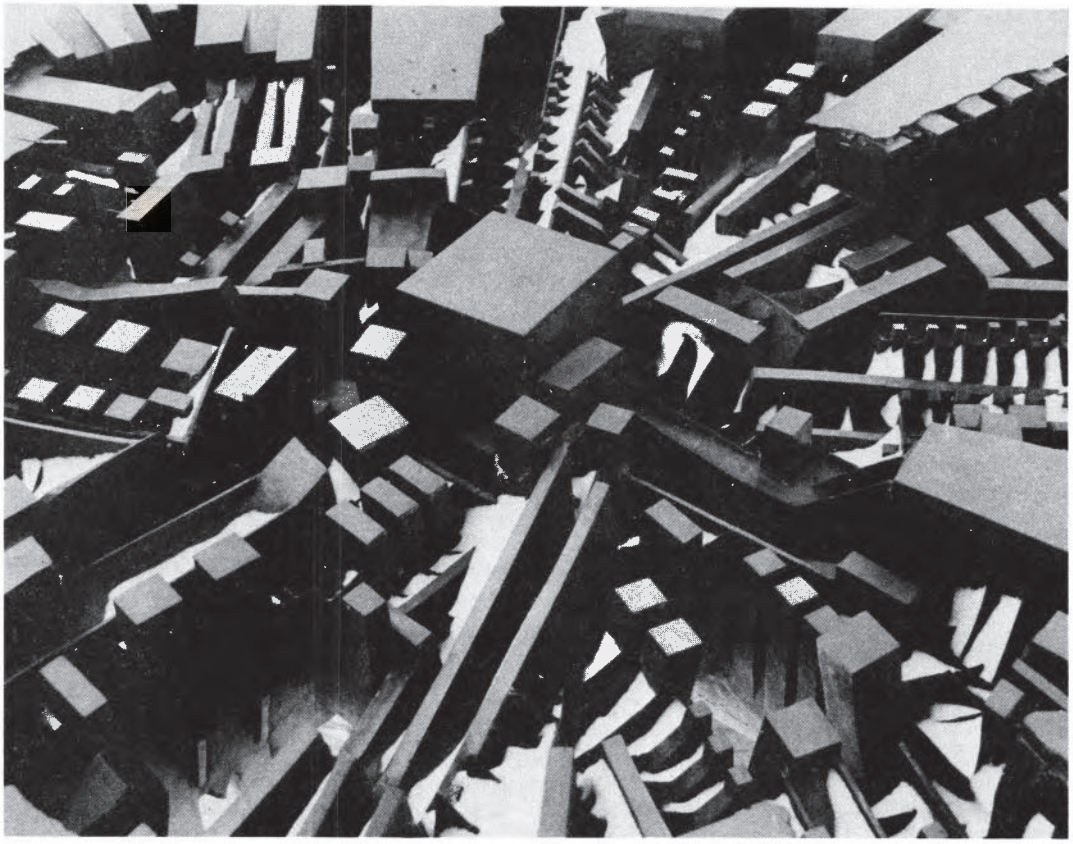
The ampersand command is a very important means to expand the power and expressiveness of Applesoft. It is used to patch in new command routines to add to the standard Applesoft command set. This trick is very popular, and you'll be pleased to learn that you don't have to write these routines yourself, but can purchase sets of added ampersand routines for convenient use. Some of these provide features which are standard in other dialects of extended BASIC and some just provide convenient ways of avoiding PEEKs and POKEs and managing other special features of the Apple.

The ampersand (&) character is treated by the Applesoft Interpreter as if it were a regular main routine command and it is assigned a Routine Token of $AF. During program execution, when this Token is encountered in the Routine Byte, Applesoft does an effective JSR to a RAM location on page three ($03F5, decimal 1013). This differs from the USR command which is called as a mathematical operator from within the FRMEVL loop (see above).

There are a variety of ampersand packages on the market, many of which were reviewed in the March 1983 issue of Call A.P.P.L.E. magazine. Both the Routine Machine from Southwestern Data Systems and Amper-Magic from Anthro-Digital provide literally scores of routines on disk and let you pick from among them a selected set to load into memory for use at any given time. Ampersoft from MicroSparc provides a limited set but is the only one to provide extra math functions including matrix multiply. Along with the Routine Machine, Ampersoft provides a high speed sorting utility. Amper Manager from Apple specializes in an extensive set of special DOS and disk sector commands.

All of these ampersand utilities suffer from the problem of requiring valuable space in RAM memory. The one exception is the Ultra ROM Board from Hollywood Hardware which provides a modest set of useful ampersand commands which reside in the Expansion ROM space (see Chapter 22) and so do not interfere with the memory requirements of any other programs. Although you get just 2K of ampersand routines in the stripped down version of the card, it can be expanded to hold 30K of ampersand routines in ROM, bank switched into the Expansion ROM space automatically when the ampersand command is issued. Since this board also includes GPLE (see above), it may be the easiest, most effective way for many programmers to enhance Applesoft.

The stripped down set of commands available from Hollywood Hardware include hex/dec and dec/hex conversion, PRINT USING which is helpful for formatting output for screen display, and string searching capabilities.



Chapter 39

Representation of Variables

Variables in Applesoft

In high level languages such as BASIC, variables are extremely important for three reasons. First, they make it easier for a programmer to think about a problem because a meaningful name can be used to symbolize a complex and changing number or string. Secondly, they provide the potential for a substantial savings in program size since a complex value can be represented just once, somewhere outside the main body of the program, no matter how many times it is mentioned within the program. Finally they provide a convenient means for several parts of a program to refer to an agreed upon location in memory where they can exchange results. So, descriptive names, potential for storage outside the main body of the program, and ease of addressing are the dominant concerns for the design of a variable system.

In Applesoft, variable names can describe locations containing four different kinds of data elements: real numbers, integers, strings and functions (equations). There are simple variables, in which each name is matched to one data element and in which the addressing scheme is very straightforward. There are also array variables, in which each name can refer to a very large number of data elements and in which the addressing scheme is a bit more involved. The amount of space in RAM memory taken up by the stored data elements, and the space taken up by the variable addressing system itself, are important concerns in planning any large programming task, and so the structure of the system and the various space requirements will be described in some detail.

Storage of Simple Variables

The Applesoft Interpreter creates Simple Variable Fields (SVFs) which are always seven bytes in length. Standard seven byte SVFs are used for description of real numbers, integers, strings and functions. These SVFs are lined up one after another in an area of RAM which the Interpreter refers to as it's variable table.

Whenever the execution loop of the Interpreter encounters the LET token, the DEF token, or an ASCII character in the Routine byte (see Chapter 38), and also, whenever the FRMEVL loop encounters an alphabetic character or the FN token in the midst of a formula it is evaluating (see Chapter 38), it assumes some variable is being referred to. Before program execution can continue, the Interpreter must first look for the appropriate SVF within the variable table and thus learn the location of the data element referred to by the variable name.

In all of these situations, control is eventually passed to some part of the PTRGET routine at $DFE3. PTRGET collects in the first two characters of the variable name. If the variable is not a function variable, it checks to see if the variable name is followed by a percent (%) sign (for integer variables) or dollar ($) sign (for string variables), and also looks for parentheses which would indicate an array variable. As long as there are no parentheses, the VSEARCH routine ($E04F) then begins scanning through the variable table to see if it can find the simple variable field with that two character variable name.

When the program is first loaded, there are no SVFs in memory; rather, they are created during the course of execution when the program is RUN. The first time any variable is encountered during program execution, VSEARCH will fail to find any SVF by that name in the variable table. When this happens, VSEARCH calls NEWVAR ($E0AC), which goes to the top end of the variable table, sees that anything important there is pushed up out of the way by seven bytes, and then sets up a new SVF with the appropriate name. All of this is done from scratch every time a program is RUN. For this reason, the very first pass through a program loop may take longer than usual because the Interpreter must repeatedly pause to set up new SVFs.

Organization of SVFs in the Variable Table

Since all of the SVFs are seven bytes in length, there is no need for a linked list format. This means that unlike a Program Line Field (see Chapter 38), an SVF has no pointer to the next SVF in the variable table. VSEARCH knows that they are all seven bytes long (see Figure 39.1); so, after locating the first entry in the table, it checks the name there, and, if it is not the SVF it is looking for, increments its own pointer by seven bytes and reads the next name, and so on until it either finds the the SVF it is looking for or comes to the end of the table empty handed and calls NEWVAR.

This is something to keep in mind when you're trying to speed up execution of your Applesoft programs. Every time a variable is mentioned, VSEARCH must scan through from the beginning of the variable table to try to find it. Variables at the beginning of the table get found sooner.

If your program mentions a variable in some loop, and that loop seems to run rather slowly, you can sometimes get some speed improvement by seeing that the offending variable is placed near the beginning of the variable table. All you have to do is to put a statement such as `XL = 0` in the very beginning of your program, and XL will be the first SVF created. It will be placed in the first position of the variable table whenever the program is RUN and will remain there as its contents are redefined later in your program. Putting it first in the variable table makes it the fastest grab for VSEARCH and hence can speed up execution of the program loop that uses it.

Variable Names in the Simple Variable Field

VSEARCH is able to distinguish between the similar variable names: "XL," "XL%," "XL$," and "FN XL" (even though the SVF for each of these has the two byte name "XL"). The four different types of simple variables (real, integer, string and function) are identified by the setting of the eighth bit in the two characters of the stored variable name. For reals, both characters have their eight bit clear, integer variables have them both set, strings are marked with the first character clear and the second set, and functions are marked with the first character set and the second character clear (see Figure 39.1).

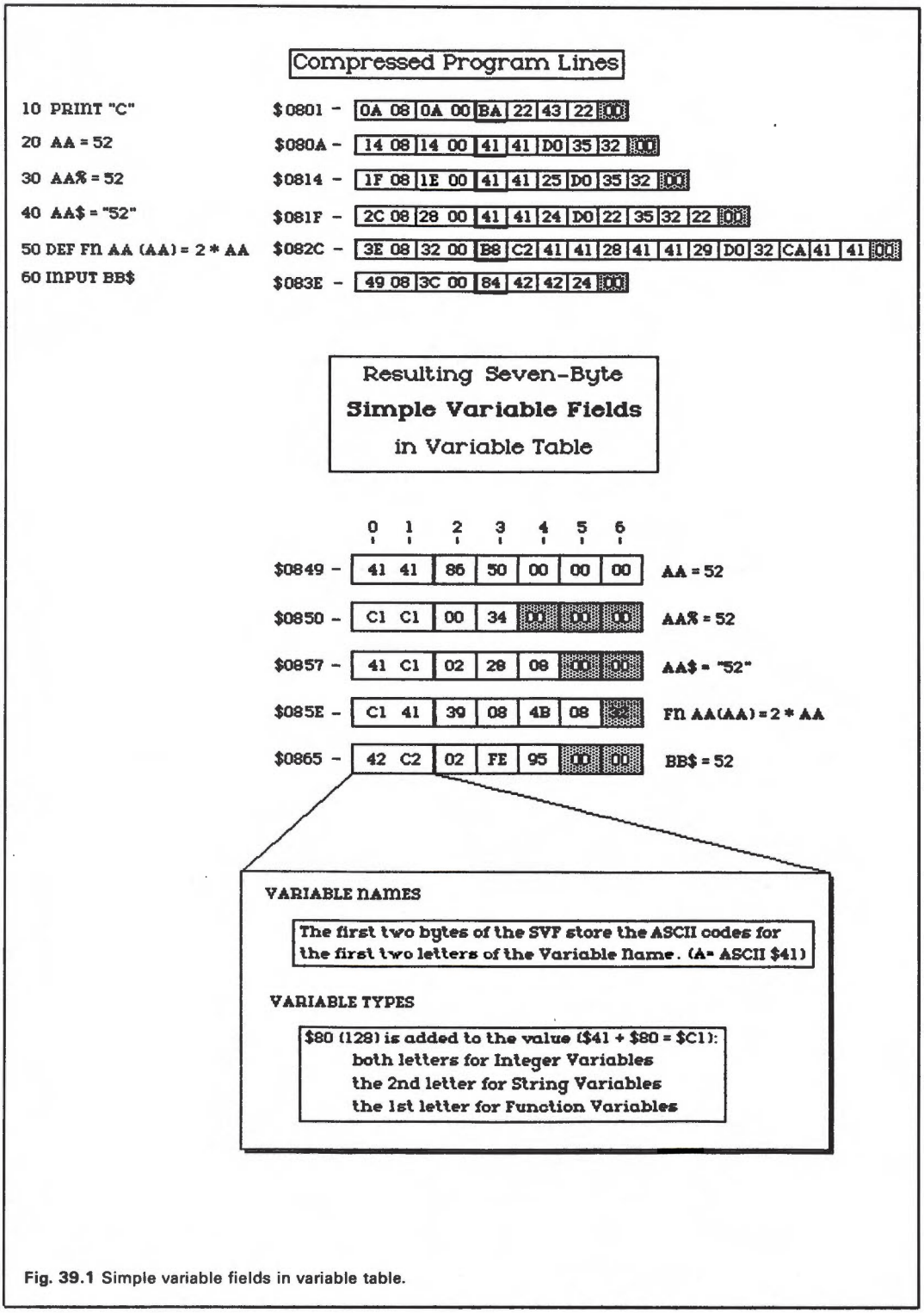


Fig. 39.1 Simple variable fields in variable table.

The ASCII code for X is $58, but when the eighth bit is set, the code becomes $D8. The ASCII code for L is $4C, but when the eighth bit is set, this becomes $CC. Therefore, the four different SVF's would be named as follows:

```
      XL    as  $58 $4C
      XL%   as  $D8 $CC
      XL$   as  $58 $CC
FN XL(      as  $D8 $4C
```

Contents of the Simple Variable Fields

The ultimate objective of the PTRGET routine is to find the address in memory at which the desired data element is stored. This is always at least a two stage process. First, the VSEARCH routine has to find the address of the appropriate simple variable field. Then, another part of PTRGET must find the address of the actual data element described by that SVF.

Real numbers and integers are stored within the SVF itself. The first two bytes of the SVF hold the variable name, but the third byte in the field is the beginning of the actual data element. To get the address of the data element, PTRGET needs only to add two to the address of the SVF. This calculated address is then delivered to the routine which called PTRGET in the first place.

Representation of Numbers in Variable Fields

Real numbers are always stored in "packed binary scientific notation," ready for use by the FAC/ARG floating point math system (see Chapter 38). In this form, each number takes up five bytes: one byte for the exponent and four bytes for the mantissa. This all fits neatly into the seven byte SVF: two bytes for the name and five bytes for the number (see Figure 39.2).

This five byte storage requirement for real numbers is true no matter what the actual value is, although numbers must be between 1.7 times 10^{-38} and 1.7 times 10^{38} (see Chapter 38). The packed format refers to the fact that the sign of the number (negative or positive) is stored in the highest bit of the four byte mantissa rather than taking up space in an additional byte as it does in the FAC. The sign of the exponent is stored in the eighth bit of the exponent byte just as in the FAC.

An Applesoft integer is stored in two bytes. The highest bit determines the sign (positive or negative) of the number, and the remaining 15 bits can describe any number between 0 and 32,767. When the eighth bit is set, the numbers are stored in two's complement form. In theory, this special storage form should simplify binary subtraction of integers. In fact, Applesoft does not do math with integers.

Whenever an integer shows up in the FRMEVL loop, the first thing that happens is that it is converted into a floating point number in expanded binary scientific notation. This even happens if no math has to be done on it. If the number has to be stored in an integer variable SVF, it has to be converted back into an integer for storage. This means that it is slower to do math on integers than it is to do math on floating point numbers. When Applesoft is asked to add 1 + 1, instead of zipping it through the 6502's ALU in a few millionths of a second, it converts it, does simulated 40 bit math in its FAC/ARG system, and then reconverts it to an integer. A bug in this procedure in early IBM PC computers caused the new PC to be unable to add 1 + 1 correctly.

Not only are integers slower to process, but they don't even take up less space in the variable table. In an integer SVF, the first two bytes hold the name, and the second two bytes hold

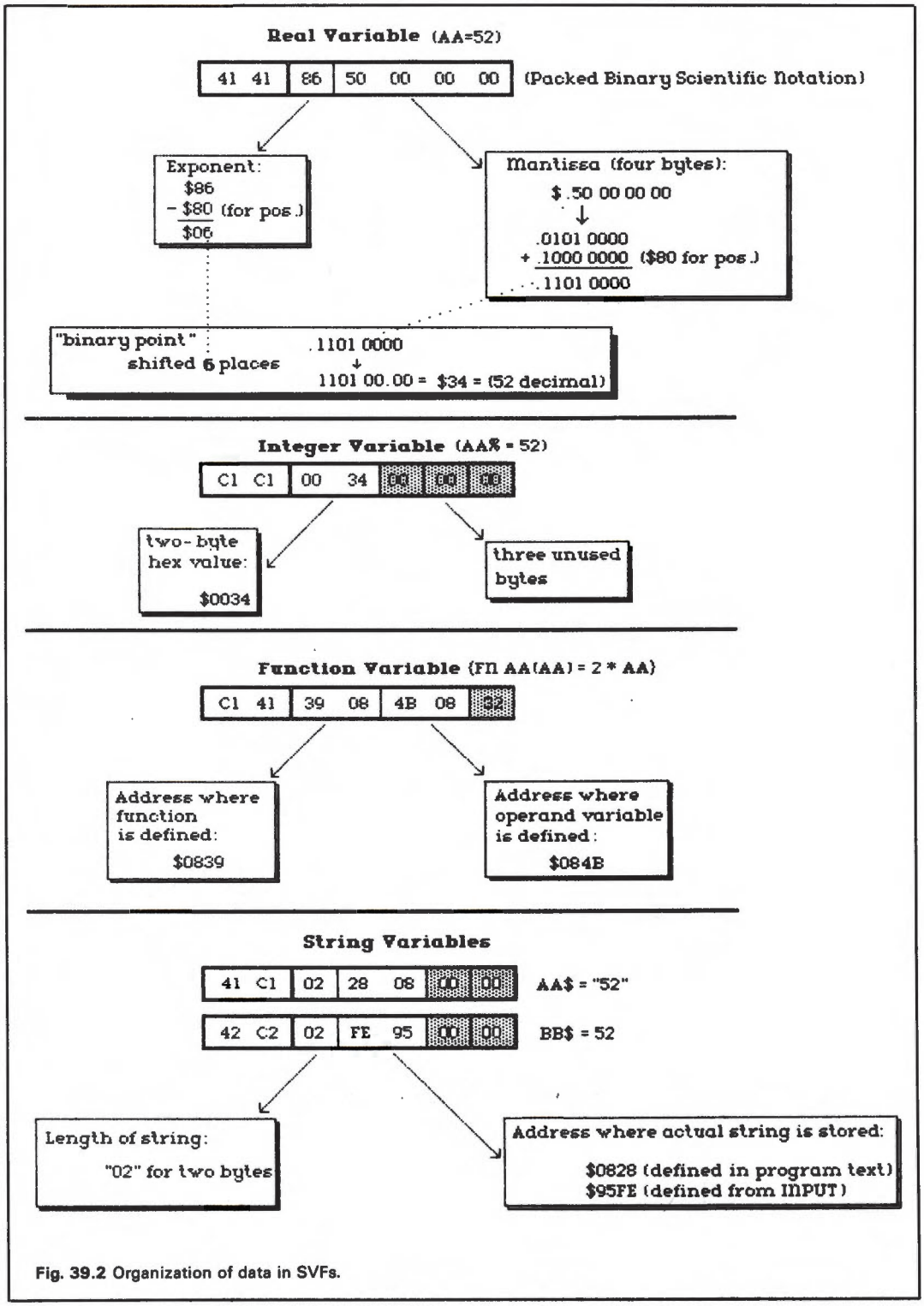


Fig. 39.2 Organization of data in SVFs.

the integer value, but the remaining three bytes are all set to zero and left unused (see Figure 39.2). There is therefore no good reason to use integer simple variables. The situation is different for integer array variables in which you reap the full space saving benefit of representing a number in two bytes instead of five (see below).

String Variables

Since strings and functions can be too long to fit into the five available bytes of the SVF, they are stored elsewhere in memory. In an SVF which describes a string, the third byte describes the length of the string (0 to 255 characters), while the fourth and fifth bytes make up a pointer which contains the address of the location in memory at which the string begins (see Figure 39.2). PTRGET can simply collect up the contents of the fourth and fifth bytes of the SVF and return that address to the calling routine.

Strings are stored in two different regions of memory depending on how they are used in the program. In many cases, a string is defined directly in some program line (i.e., 130 D6$ = "THIS IS THE STRING"). In this case, the address points into program memory to the part of the Program Line Field (i.e., the PLF for line 130 in our example) in which the string is written out.

A much more elaborate system is required for keeping track of strings which are typed in by someone using the program rather than by the programmer, as well as for reading in strings from a text file on disk (i.e., in response to INPUT A5$). These input strings are stored in a special string storage space set apart from all the other parts of the Applesoft program and variable storage areas.

The string storage space is made up of a series of String Storage Fields (SSFs) lined up one after another. These fields do not include the quotation marks, but they do include all spaces. Applesoft stores the starting address and length of each string storage field in the SVF, so the string storage fields themselves contain only the ASCII characters, and no additional information.

The SSFs are loaded into the very top of available free memory (see Chapter 40, Figure 40.2). When DOS 3.3 is in memory, the highest address available to Applesoft for string storage is $95FF (38,399). If you respond to an INPUT A$ statement by typing THIS IS, then an SSF with a length of seven bytes will be created, beginning at 38,392 ($95F8). The T will be stored at 38,392, the H at 38,393, and so on. If you then respond to an INPUT B$ statement by typing AN EXAMPLE, then a second SSF will be created with a length of 10 bytes and a starting address of 38,382 and so on (see Figure 39.3).

Updating Strings and Using the GARBAG Routine

The system for updating a string variable after it is first set up is a bit awkward. In our example, if you respond to a second INPUT A$ statement by typing A DIFFERENT STRING then a new string storage field is created with a length of 18 characters beginning in our example at 38,364 (see Figure 39.3). The pointers in the A$ SVF are readjusted to point to the new SSF, but the old string is simply left sitting where it was, abandoned in the string storage space. If you set up a program loop in which you repeatedly type in a new A$, the Apple's memory will steadily fill up with abandoned older versions of A$. The string storage space grows downward in memory in a steady march towards other important areas below it.

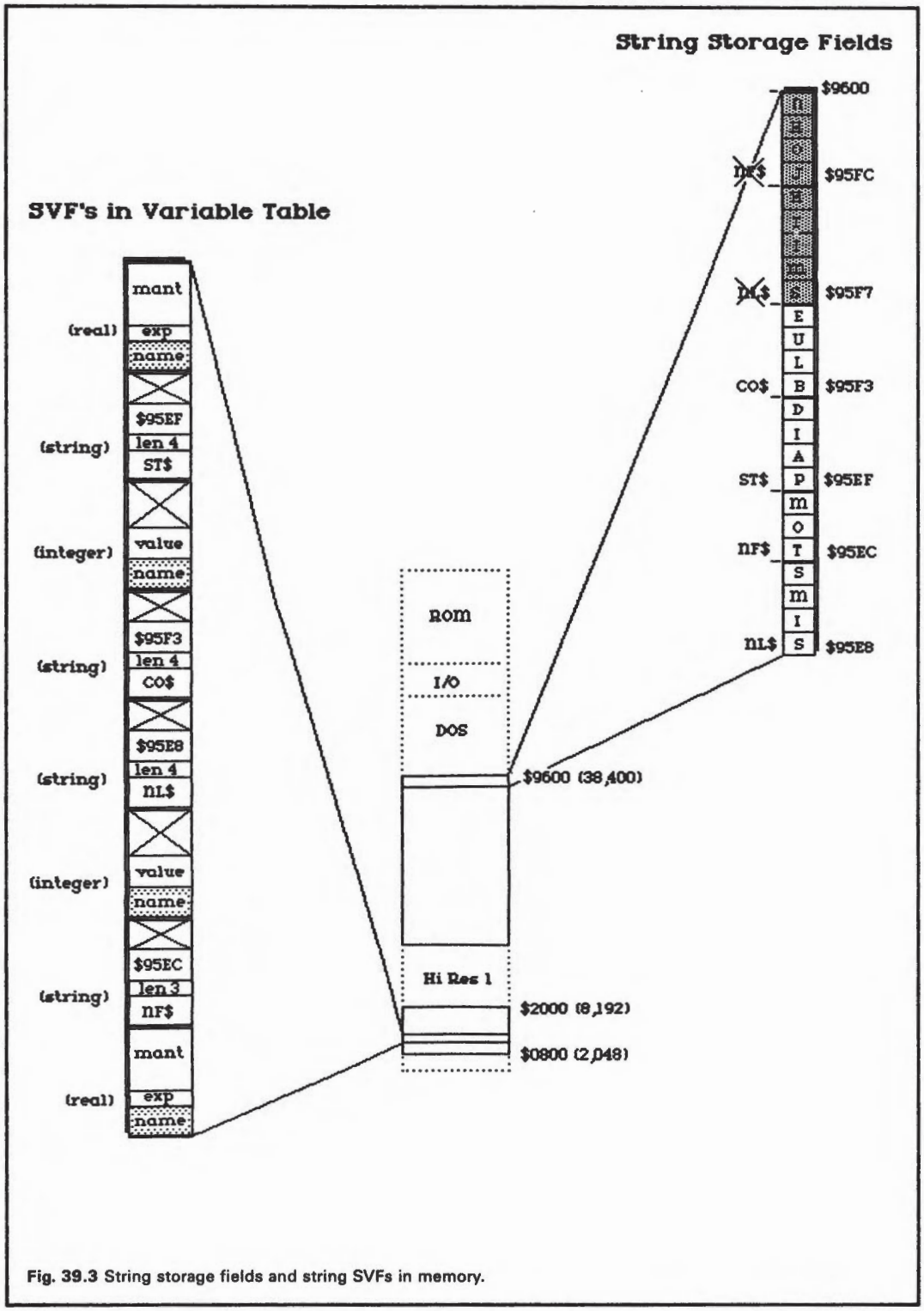


Fig. 39.3 String storage fields and string SVFs in memory.

The Interpreter has a space management system (see Chapter 40) which monitors the growth of the string storage space. When this management system detects the imminent overgrowth of the string storage space to the point where it is about to start to overwrite other important memory areas, it slams on the brakes and brings the program to an abrupt and complete halt while it tidies up the string storage space. Tidying up involves a long and tedious process formally called garbage collection.

During garbage collection, the GARBAG routine ($E484) scans through the variable table (and its array equivalent) to find the highest valid string in the string storage space. It moves this string to the top of free memory, overwriting any abandoned strings, and then resets the pointers in the SVF for the string it just moved. It then scans the variables once again for the next highest active string until all active strings have been packed neatly one after another at the top of free memory and all pointers have been reset.

Speeding Up Garbage Collection

This process can cause a rude and puzzling long pause in the midst of an otherwise healthy running program, but there are two reasonable approaches to avoiding the long delay. The first is to frequently trick the space management system into thinking that there is no more free space and thus causing it to undertake garbage collection before things get too tedious and messy in the string storage space. This is done by putting the statement X = FRE(0) at various places in your program. Everytime this statement is encountered, GARBAG will be called and program execution will halt, but if it is done frequently, there will be very little work for GARBAG to do.

The other option is to use one of the fast garbage collection routines frequently published in magazine articles (see *Call A.P.P.L.E. in Depth: All About Applesoft*), or available commercially as with the Ampersoft package by Cornelis Bongers (see Chapter 38). These routines achieve their improved performance by optimizing the algorithm used by the GARBAG routine. The principal area for improvement is in GARBAG's one string at a time approach. A very fancy routine can keep track of the addresses of all of the active strings as it makes a single pass through the variable table and its array equivalent. It then carries out all the moves and pointer changes in rapid continuous sequence without pausing for a full variable scan between each move. If you are using ProDOS, there is no problem with slow garbage collection because ProDOS automatically installs a very efficient high speed routine to do the work.

Function Variables

The storage form of function data elements is much simpler than for strings; however, the contents of a function SVF are a little more complicated than the contents of a string SVF because of the way functions are executed. This is worth a little explaining here because the new description of function variables in the *//e Applesoft Reference Manual* leaves out some crucial details which had been included in the original Applesoft Manual. These defined Applesoft functions do not behave exactly the way you would expect from your experience with the rest of Applesoft.

When you type DEF FN AB(DH) = 3 + 4 + (DH/2), you are making DH a special "dummy variable". The DEF statement tells Applesoft to respond to FN AB(X) by stuffing X into the dummy variable position on the right hand side of the equation. If you assign a value to a real variable named DH somewhere else in the program, this will have no effect on the execution of FN AB(X). The FUNCT routine ($E354) will act to trick FRMEVL into using X instead of your real DH when it evaluates the expression.

To see this odd thing taking place, try the following simple program:

```
10 DEF FN AB(DH) = 3 + 4 + DH
20 DH = 100
30 PRINT FN AB(2)
40 PRINT DH
```

When line 30 is executed, the result is nine, when line 40 is executed, the result is 100. This is important to appreciate because it means that you cannot change the action of the function in line 10 by redefining DH elsewhere in the program.

To pull this off, the SVF for a function must have two pointers. The first pointer, placed in the third and fourth bytes of the SVF, contains the address where the formula is stored in memory. This will always point to the part of program line field in which the function was first defined. The second pointer, placed in the fifth and sixth bytes, contains the address of the data element of the variable used as a dummy variable (DH in our example). Whenever a function is evaluated, the FUNCT routine can use this second pointer to do a quick switch to temporarily fool FRMEVL into using the contents of the parentheses (in the FN AB (X) statement) rather than the actual contents of the real variable by that name (use two instead of 100 in our example).

Another minor mistatement in the //e Applesoft Reference Manual is that a function takes up six bytes in memory. In fact, not only is it stored in a standard seven byte SVF, but there actually is a value stored in the seventh byte of a function SVF. This value is obscure in that it is not used for anything, but, for the record, it is the first byte of the definition of the function (see Figure 39.2).

Array Variables

The use of arrays in an Applesoft program can greatly simplify the programmer's work, and can yield substantial savings in space requirements. Within the program, an efficient structure with an array variable and some nested FOR/NEXT loops can often replace dozens of program lines. Further, when used carefully, an array can provide much denser packing of data elements in memory outside the program.

When you set up an array, you are creating an ordered address space with a series of cells for storing data elements. The structure of the array variable system gives you random access to any data element cell. Each cell within the array has an address which can be easily calculated and manipulated within an Applesoft program. Although the array address you specify must still be translated into an actual 6502 address to fetch the data element, this approach skips most of the tedium of the VSEARCH system used for simple variables and so leads to a substantial increase in speed of execution in programs which do a great deal of manipulation of data elements.

Dimensions and the Hierarchical Addressing Structure

Underlying these impressive advantages in speed, compactness and programming efficiency is what can be called a hierarchical addressing system. The space which contains the data elements is organized into regions, each region divided into sub-regions, and each sub-region divided into sub-sub-regions, and so on. To address a given data element, you specify the name of the array, the region, sub-region, sub-sub-region, etc. A brief five byte description can easily select one data element from among many thousands spread across large areas of Apple memory.

Each level in the hierarchy is called a dimension. The term dimension derives from geometric concepts used in matrix algebra. Those few in the crowd who dwell in the realm of factor analysis and multivariate statistics will be quite comfortable with the concept of a five or six dimension space. For the vast majority, however, the best bet is to forget about the geometric metaphor and think of array dimensions as levels in an address hierarchy as shown in Figure 39.5.

Array Variable Fields

An Array Variable Field (AVF) is different from a Simple Variable Field (SVF) in a number of ways. First, there is no fixed size for an AVF. The entire hierarchical address space is contained within the AVF, so that they can be very large. For instance, execution of the statement DIM X(10,10,10,5) sets up an AVF which is nearly 40,000 bytes in length.

Second, AVFs are organized into a modified linked list system within the array table. The third and fourth byte of each AVF contains a count which is based on the total number of bytes taken up by the array (0 to 65,535). This count is called an offset pointer because if it is added to the address of the array's name, the result will be the address of the next array's name. The advantage of using an offset pointer is that it does not have to be recalculated every time the array table is moved. The size of an array is determined when the DIM statement is first encountered and cannot be changed while the program is running (except by CLEAR). The offset pointer remains valid no matter where the program is stored in memory.

The third difference is that an AVF contains a description of how its space is organized. The information in this description can be used to translate an array address supplied by the programmer (such as X(4,3,7,2)) into the actual 6502 address at which the selected data element can be found.

The description is based on the DIM statement used to create the array. The first byte tells how many dimensions are in the description, and this is followed by the sizes of each of the dimensions, each in a two byte space, with the last dimension stored first (see Figure 39.4). In our example of DIM X(10,10,10,5), the number of dimensions is 4, and it is followed by $0006 $000B $000B $000B (6, 11, 11, 11). (Note that when a dimension is entered as "5" you are describing six regions: 0, 1, 2, 3, 4, and 5. Don't forget to use the space in the 0 region.)

To follow through in calculating the location of a data element in our example, you can consider what happens when FRMEVL encounters the statement X(4,3,7,2) = 18.35. The task will be to figure out exactly where to store the five byte real number representation of 18.35. The first few steps are exactly as described for simple variables (see above). The ASCII character

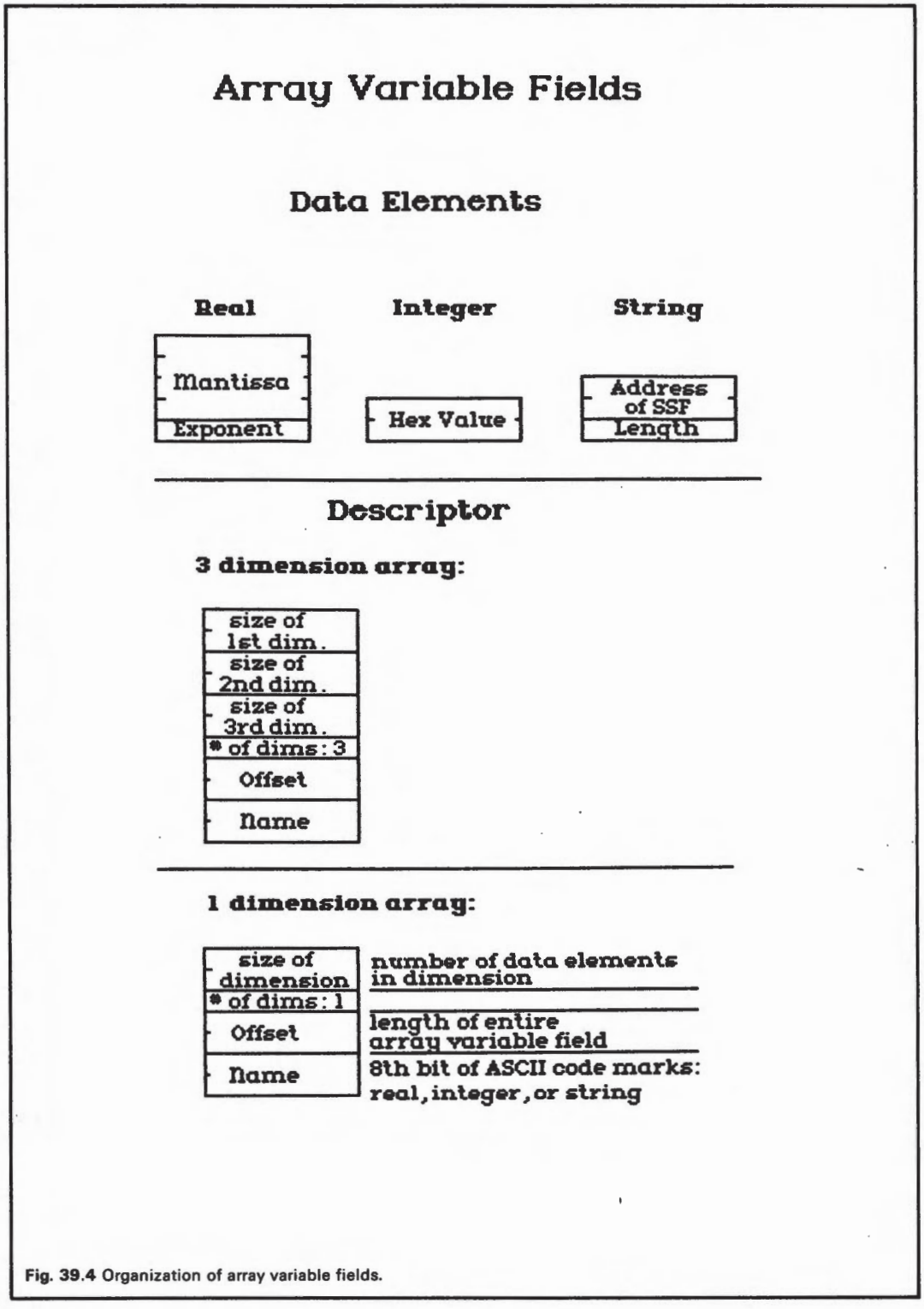


Fig. 39.4 Organization of array variable fields.

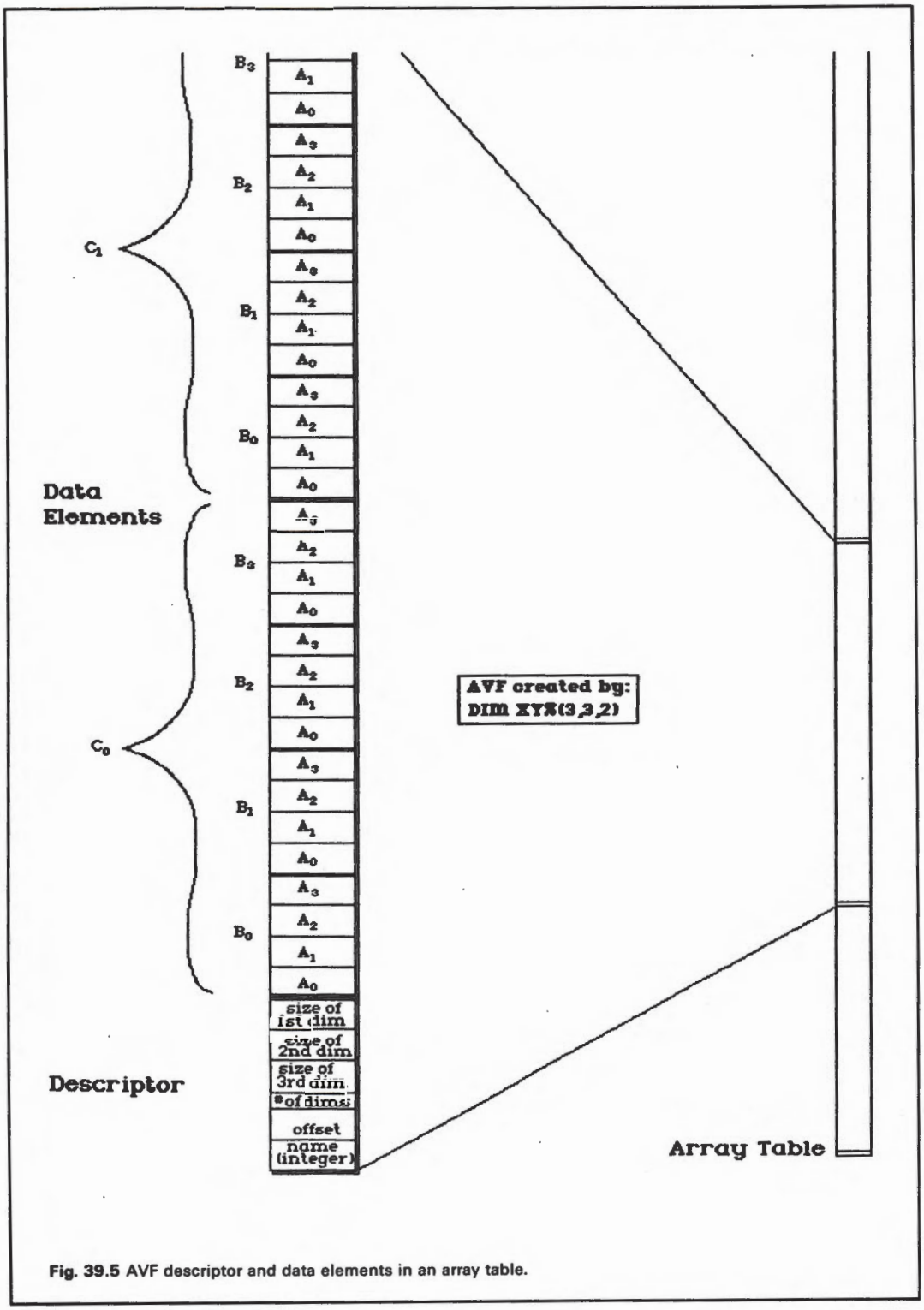


Fig. 39.5 AVF descriptor and data elements in an array table.

in the routine byte (see Chapter 38) sends control through VARL to PTRGET. In this case, PTRGET notices the parenthesis after the X and sends control out of the usual PTRGET loop to ARRAY at ($E11E), so that VSEARCH is never used.

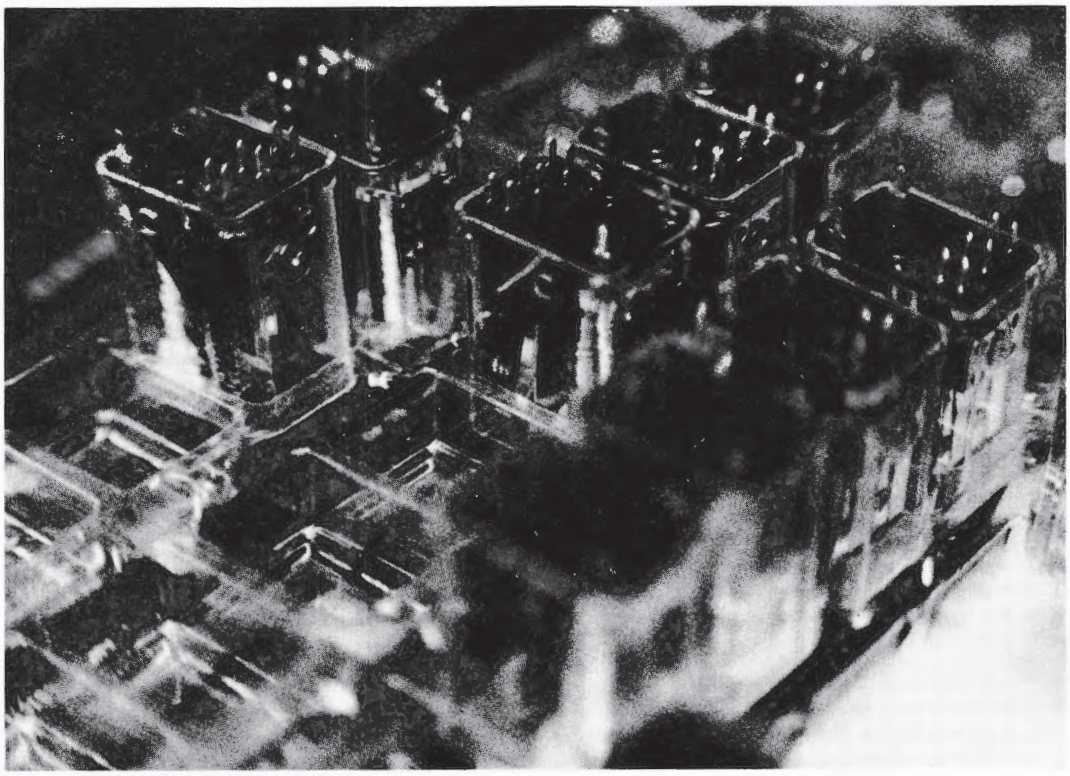
The FNDARY routine is the approximate equivalent of VSEARCH in that it locates the beginning of the array table, reads the first variable name, and, if it does not match the name it's looking for, skips on to the next AVF. VSEARCH finds the next SVF by adding seven to its current address pointer, but FNDARY must read the offset pointer and add its contents to the current address pointer. The variable names are used to distinguish between real, integer and string arrays by means of the same high bit system used for SVFs, although there is no equivalent of the function variable.

Once the proper AVF is located, the ARRAY routine can use a special 16 bit multiply routine (MULT at $E2AD) to calculate the actual position of the desired data element. In our example, the array was dimensioned as DIM X(10,10,10,5) and we are trying to carry out the statement X(4,3,7,2) = 18.35. To locate the desired data element, ARRAY must know the size of each data element, sub- sub-region, sub-region, and region. These are easily calculated from the information in the array description, and once they are known, the address of the desired data element can be calculated by advancing a pointer across the required number of regions, sub-regions, etc.

Data Elements Within the Array Variable Field

The storage format for data elements in array variables is essentially identical to the format used for simple variables: Real numbers take up five bytes, integers require two bytes, and strings are described by a length and a pointer to a string storage field (see Figure 39.2). Things are actually a little simpler in that there is no array equivalent of a function variable. The only important difference is to do with the arrangement of the data elements within the AVF.

As explained above, each AVF begins with two bytes for the variable name, followed by two bytes for the offset pointer, followed by a description of the array space which varies in size depending on how many dimensions are described (see Figure 39.4). The data area itself is made up of a long series of data elements lined up one after another, with no intervening spaces or marks. Unlike the SVFs, there are no wasted or unused bytes in an AVF data area. Integers take up just two bytes, so they provide a substantial space savings over five byte real numbers in an array. In string array variables, the three byte string descriptors are also packed in one after another, and the string storage fields they point to in the string storage space are freely intermingled with SSFs pointed to by simple string variables (see Figure 39.3).



Chapter 40

Applesoft Memory Management

The hard fact of the matter is that a 64K computer is a tight space to work in for many BASIC programs, and even when they fit they may not run fast enough to be completely useful. These problems extend to the IBM PC and the various Apple 8088 boards in which the BASIC Interpreter allows for only 64K of memory use and may run even more slowly than in the standard Apple II. The new 68000 based Applesoft Interpreters for the Saybrook and ETC Apple II plug in boards (see Chapters 30 and 41) provide for use of as much RAM as you can buy and for some increases in speed of execution, and there are a number of other, less expensive, hardware and software solutions. In all cases, though, the first step is to know your enemy.

There are a variety of reasons why Applesoft programs run out of space or run slowly, and many of these can be handled with inexpensive software fixes. Further, if your problem requires a hardware solution, you should be clear on exactly what hardware to invest in. To explore the various sources of trouble, it will be handy to run through a quick review of some of the information presented in Chapters 38 and 39. If the following review is not enough for you, but you're not up for plowing through all of the preceeding 50 odd pages, you should read *Applesoft from Bottom to Top* by Val Golding in *Call A.P.P.L.E. in Depth: All About Applesoft* which is a thorough but much more concise coverage of the same material.

Space for the Four Parts of a Running Applesoft Program

There are four parts of a running Applesoft program, each of which behaves in its own unique manner as the program builds and runs. The Applesoft Interpreter has a standard method for positioning these four parts within available Apple memory, but these standard positions aren't always the best. Some space problems actually result from the positioning of the program parts rather than their absolute size per se, so the Interpreter also permits you to choose where each part will be placed.

The first and most obvious portion is the program line space which contains the tokenized and compressed lines of the BASIC program you typed in (see Chapter 38). This part grows as you type in new lines, but stays fixed in size while the program is running (see Figures 40.1a and 40.1b). As soon as the program is RUN, however, the second, third, and fourth parts begin to grow.

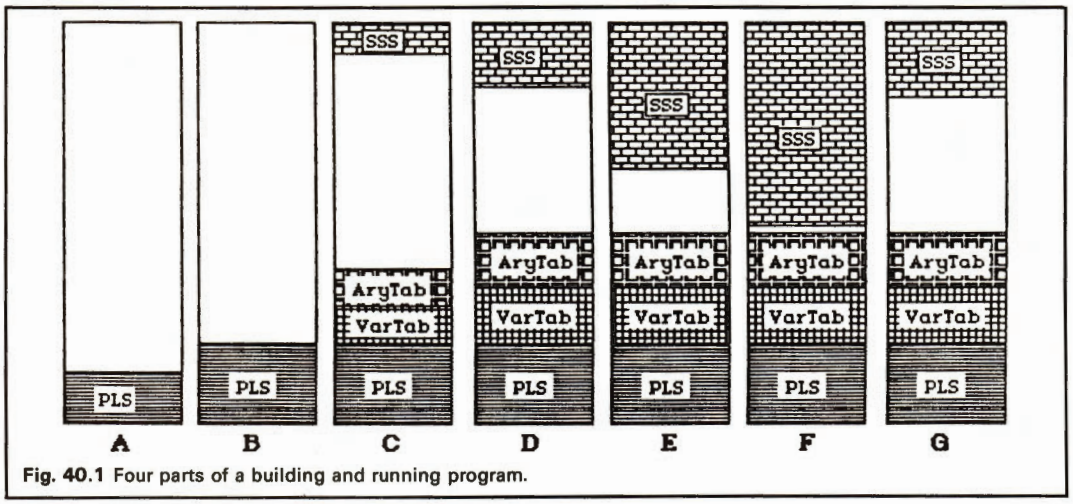


Fig. 40.1 Four parts of a building and running program.

The second part is the variable table (see Chapter 39), which contains a seven byte entry called a Simple Variable Field (SVF) for every simple variable mentioned in the program. The SVFs are always seven bytes, no matter whether they describe a real number, integer, string or function. They are added to the variable table in the order in which the variable names are encountered by the execution routines of the Applesoft Interpreter. This table continues to grow until all the simple variables in your program have been encountered at least once (see Figures 40.1c and 40.1d).

The third part is the array table (see Chapter 39) which contains a large block of reserved memory space called an Array Variable Field (AVF) for each array variable named or created by a DIM statement. Each AVF has a housekeeping section which is seven bytes long for an array with one dimension, nine bytes long for two dimensions, 11 bytes for three dimensions and so on (see Chapter 39, Figure 39.4). The AVF also includes a data area whose size depends on the type of variable and on the size of the array defined in the DIM statement.

The number of cells in the array can be found by multiplying the dimensions by each other. The size of each cell depends on which kind of data element is being used. Real number cells take up five bytes each, integer cells take up two bytes, and string cells have a three byte descriptor which points to the string's actual storage location elsewhere (see Chapter 39, Figure 39.2). When the Interpreter encounters the statement DIM X(10,10,10,5), it creates an enormous 40,000 byte AVF, with 13 bytes of housekeeping information and the remainder all cleared to zero and reserved for future references to that variable.

The AVFs are set up as the Applesoft Interpreter encounters DIM statements or the names of small (less than 10 cells) undimensioned array variables during program execution. Careful use of arrays can be more space efficient than the use of simple variables, but it's very easy to block out huge ranges of memory.

The fourth space is made up of String Storage Fields (SSFs) which contain the strings you assign to simple and array string variables. The SVFs and AVFs don't actually contain the strings, but rather hold the addresses of SSFs in the string storage space (see Chapter 39, Figure 39.3). Strings typed into program lines while the program is being built stay where they are, and need no SSFs, however, all strings created by string manipulations or entered by a user in response to INPUT statements are stored in the string storage space.

The string storage space grows as new strings are typed in while the program is running. One odd feature is that when you assign a new string to a variable name, (i.e., change from A$ = "THIS" to A$ = "THAT"), the old abandoned SSF is left sitting up there in the string storage space wasting RAM. As string reassignments proceed, the string storage space just keeps growing and growing until there is no free space left in the Apple (see Figures 40.1c, d, e, and f). At this point, your program grinds to a halt and the Applesoft Interpreter goes through a long and tedious process of removing all the abandoned strings and compacting all the active strings into the smallest possible area (see Figure 40.1g).

Available Space in RAM

The program line space is loaded into memory beginning with the lowest line number at the lowest available address in RAM, and filling on up to its full size as soon as it is loaded. The variable table is usually begun immediately above the last program line, and the array table begins just above the last SVF in the variable table. Whenever the Applesoft Interpreter encounters the name of a simple variable, it pushes the array table out of the way by seven bytes to make room for a new SVF at the top of the variable table (see Figures 40.1c and d). As DIM statements are encountered, new AVFs are added at the top of the array table. The string storage space is begun at the highest available address in RAM and builds downwards as new SSFs are created.

In a wide open, unrestricted, 64K memory space, the program line space, variable table, and array table would be stacked one on top of the other all starting from the lowest address in memory and rising upwards. String storage space would begin at the very top of high memory and build downwards as it filled with strings, and you would have no problems until the strings growing down from the top smashed into the rest of the program growing up from the bottom (see Figure 40.1f).

The Apple II, however, does not have a wide open, unrestricted 64K memory space. The layout of the Apple's memory is described in detail in Chapters 21 and 26, but to review briefly, there are several large swaths chopped out of the memory range. The Applesoft Interpreter is aware of some of these and tries to forbid you from using them. Others are ignored by the Interpreter and cause program disasters when conflicts arise.

The bottom end of the memory range is reserved for deep internal operations of the 6502 and for storing the characters which appear on the standard text display on the video screen. The various requirements at the low end of memory chop off the first 2K and this range of memory is almost universally kept off limits to Applesoft programs (see Chapter 21, Figure 21.4).

The limitations at the high end are much more severe. The Applesoft Interpreter is quite convinced that you cannot put more than 48K of RAM in an Apple. Even if you have a //e or //c, or a 64K II/II+, the Interpreter cannot be convinced to make use of the high 16K without significant intervention on your part.

It's not that the high 16K of address space is completely wasted. In fact, it is taken up by 2K of ROM containing built-in input/output routines for the screen and keyboard, 10K of ROM for the Applesoft Interpreter itself, and 4K reserved for routines to operate peripheral cards.

There are ways of doing fancy bank switching to get access to the high 16K of RAM, and there are also ways of using extra RAM on the //e auxiliary card, but the native Applesoft Interpreter

as shipped cannot be made to do this. There are, however, commercially available utility programs which will let you use extra RAM. Further, ProDOS offers a convenient means of using the 64K of auxiliary RAM in a //e or //c.

An additional limitation on the high end is that space must be set aside for DOS 3.3 to be loaded into RAM. If you want to be able to save your program on disk or to use disk based text files, DOS 3.3 must be loaded, and it takes up an additional 10K at the top of memory. ProDOS is even larger. It requires 25K in its intial release and may get larger as it is enhanced. ProDOS is stored partly in the bank switched high 16K of RAM, but then extends down into the main 48K of RAM by about 10K. Unlike DOS 3.3, this is not a fixed bottom limit and more memory can get gobbled up by ProDOS as the BASIC program runs.

As a result of the limitations on the bottom and on the top, most Applesoft programmers are limited to a 36K space between 2,048 ($0800) on the bottom and 38,400 ($9600) on the top (see Figure 40.2a). There are a variety of utility programs called DOS movers (i.e., Beagle Brothers Pronto-DOS) which can move DOS 3.3 up into the high 16K of bank switched RAM (see Figure 40.4f). This gives you a much improved 46K to work in.

Reserved Areas within the Applesoft Memory Space

The final space limitation has to do with two large indigestible lumps of reserved memory sitting in the midst of the available RAM space. These are the two areas of display memory used for setting up high resolution graphics displays. The Apple has a video generator system (see Chapters 5, 6 and 7) which works independently of the 6502 to control video output. The Applesoft Interpreter communicates with the video display generator by loading bit patterns into RAM in an agreed upon area of memory. The video display generator then scans that area of memory sixty times a second to find out what to display on the screen.

There are actually four areas of RAM which can serve as display memory, although the video display generator only uses one at any given time. The first of these has already been mentioned. It is used for text display, takes up just about 1K of RAM, and is stowed safely away below the 2K bottom line. There is a second text area, between 2K and 3K (see Chapter 21, Figure 21.4) but it is rarely used for display and isn't a concern for most BASIC programmers.

The real problem is for the high res graphics display memory areas. To begin with, these graphics areas are much larger than the text areas. The reason for this is that to control the text display, the Apple only has to worry about putting characters into 40 columns × 24 rows = 960 character positions. For high res graphics, however, the Apple must be able to address a much higher number of individual bit positions. The detailed layout of the high res display areas is explained in Chapter 21, but the result is a requirement of 8K for each area.

There are two graphics screens, one between 8,192 ($2000) and 16,383 ($3FFF) and a second between 16,384 ($4000) and 24,575 ($5FFF), as shown in Figure 40.2. If your program does not use high resolution graphics, then you can ignore the presence of these two display areas. There is nothing special about the RAM itself and it can be used for any purpose. However, if you have an HGR or HGR2 statement in your program (or use the high res screens through some other nefarious route) you must keep all parts of your program out of the way of the graphics system or else meet with certain disaster.

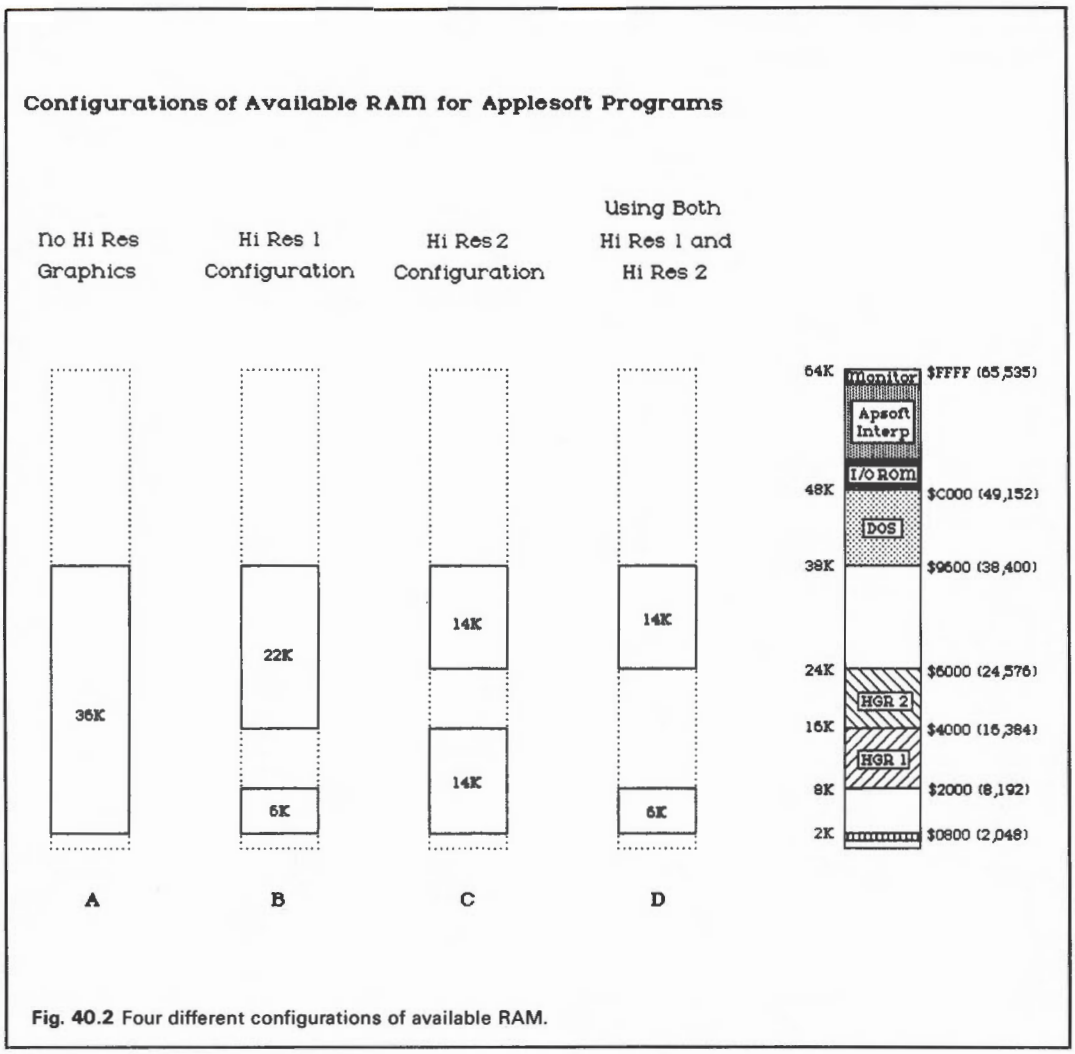


Fig. 40.2 Four different configurations of available RAM.

Use of Available Space by Applesoft Programs

As a result of the positioning of the high res graphics screen memory areas, there are four possible memory configurations for running Applesoft programs (see Figure 40.2). The most straightforward of these applies only to programs which do not use high res graphics at all. Programs with no graphics operate in a wide open 36K memory range, and when this kind of program runs into space trouble, it requires some sort of modification as described below.

If your program uses one, the other, or both of the high res screens, however, you are restricted to using memory which is not reserved for your graphics displays. Each screen area is a solid 8K block, and the two 8K blocks cannot be moved from their assigned positions because the video display generator can't be refocused to read data in other areas of RAM. Programs which use both high res one and high res two must, of course, set aside a full 16K for the display (Figure 40.2d).

Effects of Graphics/Program Memory Conflicts

This is a no compromise sort of thing. If any part of your program extends into the memory space between 8,192 and 16,383 ($2000 and $3FFF) that part of your program will be irretrievably erased when the Interpreter encounters and executes your HGR command. As far as the Interpreter knows, it is just clearing junk out of the display area.

If your program line space extends into the high res area (see Figure 40.3b) and an HGR command is executed, the offending part of your program is simply lopped off. The Interpreter won't actually discover this until it tries to do a GOSUB or GOTO into the missing area, misses the NEXT in a FOR/NEXT loop, or just abruptly steps off the edge of your truncated program. Going over the edge simply ends the program if there are a couple of $00s there, but otherwise you get a Syntax Error or some other bizarre result.

The simplest way to find out if this has happened is to LIST the program without reloading it; the end of your program will have vanished. Another way to test for this is to type HGR before you load your program. You can watch the screen as DOS places the program in memory, and if it is extending up into the graphics area you will see it as scattered patches of dots on the graphics screen which increase in number until the load is finished.

The effects are a bit more subtle if your variable table or array table is getting cleared by the HGR command (see Figure 40.3c and d). When a simple variable gets wiped out, the Applesoft Interpreter will simply create a new SVF the next time it encounters the variable name again in the program; however, any data stored in the original SVF will be lost. The effect on large arrays is more obvious; if an AVF is erased, the next reference to that variable will cause a Bad Subscript or Out of Memory error. Since Applesoft can't find your array, it assumes you never created the AVF with a DIM statement. Smaller arrays (less than 11 elements in each of three dimensions) are dimensioned automatically by Applesoft without a DIM statement, so these will just get new AVFs although the old data will be gone.

If you are actually viewing the graphics display screen while this havoc is being wreaked, and the damage is not severe enough to actually make the program stop running. You will see the Interpreter's attempts to rewrite the SVFs and AVFs as the appearance of odd patterns of dots and spaces on the graphics screen.

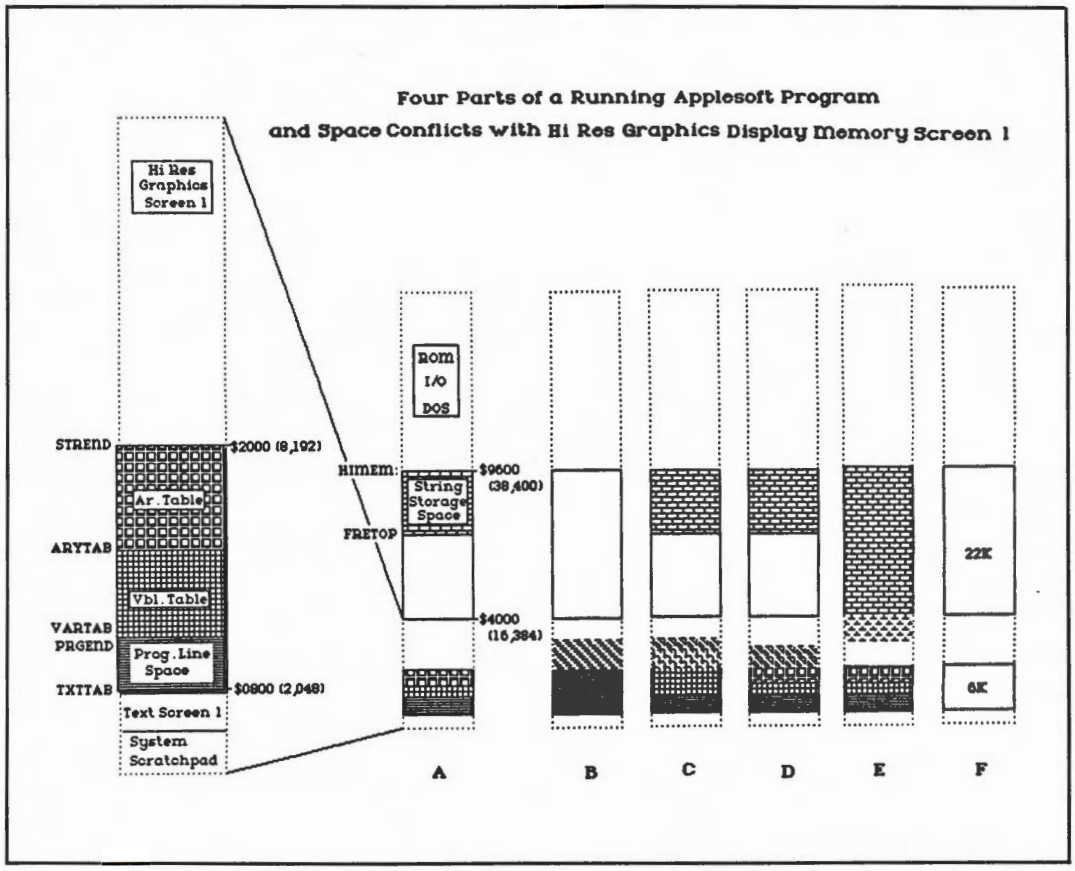


Fig. 40.3 Space conflicts between program parts and hi-res graphics screen 1 memory.

Finally, if the string storage space grows down into one of the graphics areas (see Figure 40.3e), the results get much more strange and confusing. The program will run along just fine. There will be no error messages and no missed cues. However, some of your strings will print out on the screen with bizarre and garbled contents. This is because some of the string storage fields pointed to by string SVFs and AVFs now contain whatever bit patterns happen to be on the graphics screen rather than the ASCII codes you put there. If you enter new strings, strange dots will appear on the graphics screen (bit patterns of your ASCII codes), but your strings will now print out correctly until you do more plotting or issue another offending HGR2 or HGR statement.

The Program Size Parameters

If your program is showing some of the symptoms mentioned in the preceeding paragraphs and you want to get a clear idea of what is going on, it's actually fairly easy to get an exact measure of the size of your four program parts at any time, even while the program is running. This is possible because the Applesoft Interpreter keeps careful track of the beginning and ending address of each of the four parts. These values are stored in standard locations which you can always get a look at by using a few PEEK commands. For those who aren't interested in fiddling with PEEKs and hexadecimal numbers, there is a subroutine shown in Listing 40.1 which will do all the work for you.

Most of these Program Size Parameters (PSPs) are stored in a tight little clump between $0067 and $0075 (decimal 103 and 117) with one more value stored up at $00AF/$00B0 (175,176). The memory location in which each PSP is stored as shown in Table 40.1 and the program positions they point to are shown in Figure 40.3.

TXTTAB points to the beginning of the program line space ("program text table"), and PRGEND points to the end of the program line space. The beginning of the variable table is stored in VARTAB, and the beginning of the array table is stored in ARYTAB. The top of the pile made up by the program line space, the variable table, and the array table is called STREND ("storage end"). The highest value in RAM which is available for use by Applesoft is stored in MEMSIZ, and the string storage space builds down from there to FRETOP.

When the Applesoft Interpreter first wakes up, it stores the value $0800 (2,048) in TXTTAB. This assures that any program lines typed in by the user or loaded in by DOS will keep clear of the reserved areas in the first 2K of memory. Once a program is fully loaded and you have finished typing in program lines, PRGEND can be set to point to the end of the program. When the program is RUN, the Applesoft Interpreter usually tries to begin the variable table just a few bytes beyond the end of the program, and to place the array table just above the variable table. To do this, it sets the VARTAB value to point to a location just above PRGEND, and it sets ARYTAB to point to a location just above VARTAB (see Figure 40.3).

As the program runs, the necessary SVFs and AVFs are created, and the positions pointed to by ARYTAB and STREND increase steadily. Once all simple variables have been mentioned and all arrays have been dimensioned, STREND reaches a permanent maximum size.

The string storage space will grow every time a string is created by string manipulations (LEFT$, MID$, etc.), entered in response to an INPUT or GET statement, and also every time the contents of an existing string is changed (see Chaper 39). In most programs, FRETOP never reaches an absolute final value. Rather, it moves downward towards STREND, and

when the two meet, program execution halts and Applesoft cleans out abandoned versions of strings, compacts everything, and moves FRETOP back up aways for more work, a process called garbage collection.

Program Size Check Routine

```
 10  PRINT CHR$(12): HOME
100  PRINT "This line represents the rest of your program"
110  VTAB 5: HTAB 3 : PRINT "*** TO RUN THE SIZE CHECK, JUST TYPE 'PSP'"
120  VTAB 10 : INPUT "";A$
130  IF A$ = "PSP" THEN GOSUB 9500 : GOTO 10
140  END


9500  REM ************************************************************
9510  REM *********          MEMORY CHECKING ROUTINE          *********
9520  REM ************************************************************
9530  MA = 101
9540  FOR MC = 1 TO 7
9550  MA = MA + 2
9560  IF MC = 6 THEN MA = 115
9570  IF MC = 7 THEN MA = 175
9580  MM(MC) = PEEK (MA) + PEEK (MA + 1) * 256
9590  NEXT
9600  PRINT CHR$(12): HOME: VTAB 3
9610  PRINT          "TXTTAB = "MM(1)"  PRGEND = "MM(7)"  VARTAB = "MM(2)
9620  PRINT : PRINT "ARYTAB = "MM(3)"  STREND = "MM(4)"  FRETOP = "MM(5)
9630  PRINT : PRINT "HGR2BG = 16384  HGR2EN = 24576  HIMEM = "MM(6)
9640  VTAB 14: PRINT "  The variable table is : "(MM(3) - MM(2))" bytes long."
9650  PRINT: PRINT "  The array table is : "(MM(4) - MM(3))" bytes long."
9660  PRINT: PRINT "  String data stored so far takes up : "(MM(6) - MM(5))" bytes."
9670  PRINT: PRINT "  The remaining space for data is : "(MM(5) - MM(4))" bytes."
9680  PRINT: PRINT "  HIT ANY KEY TO CONTINUE"
9690  GET MM$
9700  RETURN
```

Listing 40.1 Program size check routine. If you add this routine to the end of your program, you can get a quick check on program size. It creates a few variables and adds a little length, but it is easier than doing all the PEEKs yourself.

Program Size Parameters

Storage Location for Pointer			
Hex	Dec		
$B0 $AF	176 175	PRGEND	(Top end of program text, usually just below VARTAB)
$74 $73	116 115	MEMSIZ	(Top end of string storage space, ↑w/DOS mover, ↓ w/HIMEM:)
$70 $6F	112 111	FRETOP	(Bottom end of string storage space, current top of unused RAM)
$6E $6D	110 109	STREND	(Storage End; top of your program, simple variables, and arrays)
$6B $6C	108 107	ARYTAB	(Beginning of Array Table)
$6A $69	106 105	VARTAB	(Beginning of Variable Table, modify with LOMEM: command)
$68 $67	104 103	TXTTAB	(Beginning of program text, usually at $0800, modify with POKEs)

To read PRGEND type:

```
PRINT (PEEK (175)) + ((PEEK (176)) * 256)
```

To set TXTTAB to 16385 (just above the top of HGR screen memory)

1. Generate the page number:

```
PG = INT(16385/256)
PG = 64
```

2. Find offset within page:

```
ST = 16385 - (256 * PG)
ST = 01
```

3. POKE the page number into the high byte, and the offset into the low byte

```
POKE 104,PG
POKE 103,ST
```

Table 40.1 Program size parameters. Whenever your program parts change size, these seven pointers are adjusted to keep track. Each takes up two bytes in the zero page of Apple memory. You can read them with PEEKs or use the size check routine in Listing 40.1. The programs in Listings 40.2 and 40.3 reaim some of the pointers.

Reading the Program Size Parameters

The best time to check on the PSPs, or to use the size check subroutine from Listing 40.1, is after the program has been through enough of its loops that all simple variables and array variables are encountered at least once. Of course, if your program has been crashing for reasons you suspect have to do with space conflicts, a quick check just before the crash should give you a pretty clear idea of what has been happening. To monitor the size changes of the string area, you may want to do more frequent checks.

To use the size check subroutine, you can include menu options at various points in your program which let you do a GOSUB to it, or you can stop the program with a Control-C and then type an appropriate GOTO in response to the square bracket prompt. Note that this routine sets up some of its own variables and arrays, so remember to avoid variable name conflicts, and keep in mind that running the subroutine will cause your variable tables to grow a little.

Using Program Size Parameters to Avoid Conflicts with HGR

It may be helpful if you start this project by drawing out your particular memory configuration on graph paper, and using paper models of your four program parts (program line space, variable table, array table, string storage space) to figure out where each fits best. Of the four possible memory configurations shown in Figure 40.2, the HGR configuration (see Figure 40.2b) is most commonly the one causing trouble. The advantage of using HGR rather than HGR2 is that it is much easier to set up a mixed text and graphics screen when you are using HGR. Only very skilled programmers can manage mixed text and graphics with HGR2. In addition, //e and //c users must turn off the 80 column text system with Escape, CTRL-Q in order to permit the HGR2 command to work and get access to HGR2 display memory.

LOMEM: To Move the Variable Tables

Assuming that you are using HGR, then the cause of the problem is obvious. There is only 6K available for your program line space, variable table and array table. If your program text is settled near its final form and takes up only 4K or 5K, then your problem is with the size of the variable tables and there is a wonderfully simple solution. This is the one memory problem for which Apple has provided an easily used command to resolve the space conflict. All you have to is to put the statement LOMEM: 16384 into the very first line of your program (i.e., 10 LOMEM: 16384).

The first thing that happens when you run the program is that the LOMEM routine causes the hexadecimal equivalent of 16,384 ($4000) to be stuffed into VARTAB. This must be done before any variables are mentioned in your program so the safest place for it is as the very first line of your program. Once the Interpreter has executed this command, it will proceed through the rest of your program in the normal fashion, except that it will begin to build the variable tables beginning at address 16,384 instead of beginning immediately at the end of your program line space. The address 16,384 is chosen because that is the first address above the top of the high res one screen memory area. The results of this maneuver are shown in Figure 40.4b.

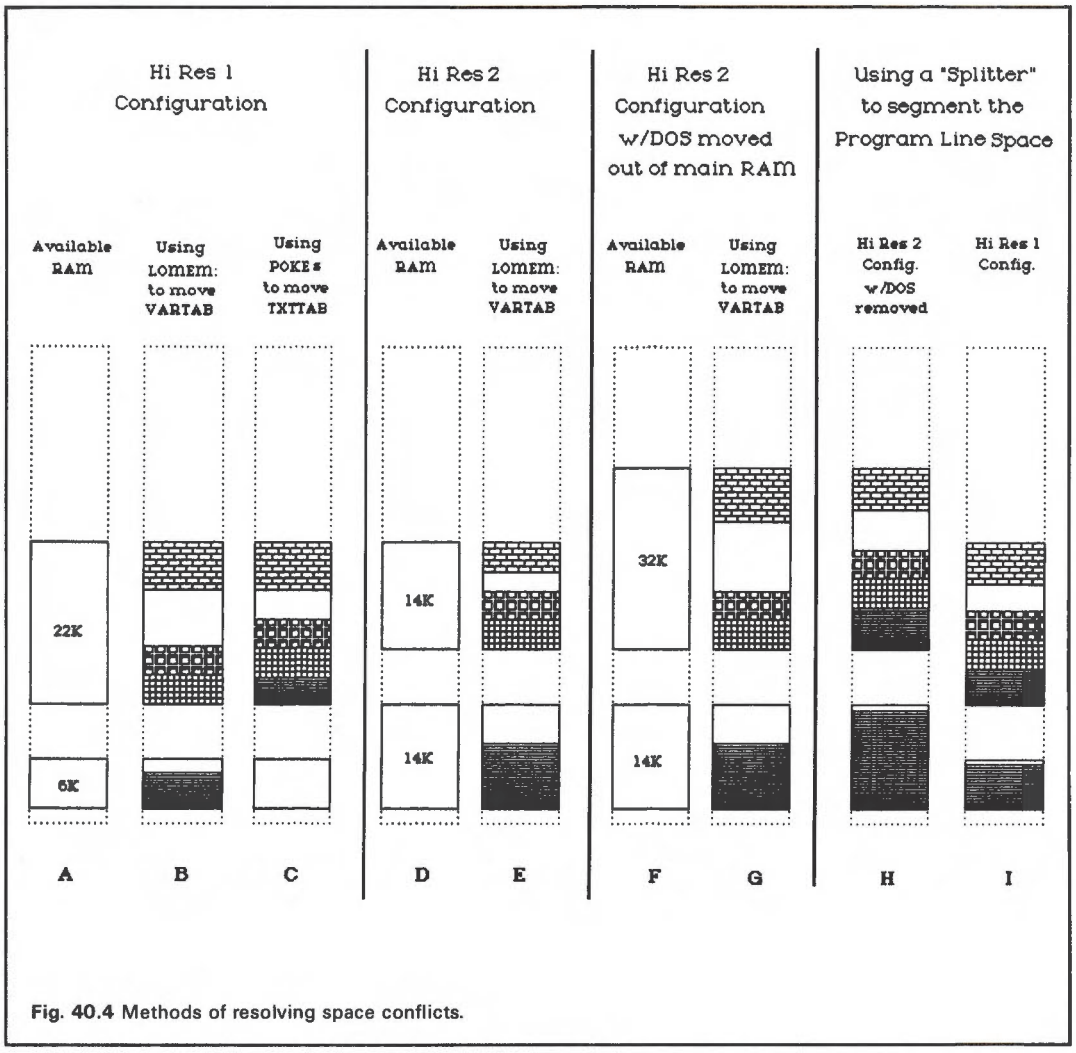


Fig. 40.4 Methods of resolving space conflicts.

The LOMEM: 16384 command gives you a full 22K for your variables, arrays and strings, but it only works as long as the program itself fits completely into the 6K below the high res one screen memory area (see Figure 40.4b). If your program text itself is longer than 6K, you have two options. The first is to further compress your program text (see below), but this is a bit of a bother, and is only necessary if your variables and strings are already taking up more than 16K in the space above the high res one screen. In most cases, you can use a second, much simpler option.

Changing TXTTAB to Move the Whole Program

Just as VARTAB could be changed to push up the base of the variable tables, you can also change TXTTAB to push the base of your program up above the high res area. This is a little trickier because there is no built in command for doing this and TXTTAB must be modified *before* your program is loaded. To reset TXTTAB to point above high res one, you should respond to the square bracket prompt by typing the four following statements, each followed by a Return:

```
POKE 104,64
POKE 103,1
POKE 16384,0
NEW
```

This should be followed by your LOAD or RUN command. This procedure will not work if your program is already loaded. Also, if you had previously put a LOMEM: statement in the program, you should be careful to remove that statement before you start to use this TXTTAB change.

What you have just done is to directly shove a new number into each of the two bytes of TXTTAB. When DOS or the Interpreter looks at TXTTAB it will see the hexadecimal equivalent of the two numbers you placed there which it will read as $4001 (since 64 decimal = $40 hex, see Table 40.1). DOS responds to this by loading in the program to start at $4001 (16,385) instead of loading it at the normal start of $0801 (2,049). The effect of this trick is shown in Figure 40.4c.

The third POKE puts a $00 at the beginning of your program which the Interpreter requires as a start-of-program marker. The NEW command tells the Interpreter to readjust several other built in parameters to adjust to the unusual starting location of your program.

The two problems with this method are that it is a bother to have to type in three POKEs every time you load your program and you could never hope to get some other user to type in the POKEs. The easy way out of this is to use an EXEC file to do the POKEs and to load or run your program. A brief program is presented in Listing 40.2 which will create the necessary EXEC file. Once this program is run and the EXEC file is stored on your program disk, it can be run by typing EXEC HGR LOADER and hitting Return. If you want everything to be completely automatic you can also set up your "Hello" program to run the EXEC file automatically.

TXTTAB EXEC MAKER

```
 10  D$ = CHR$(4)
 20  HOME : PRINT CHR$(12): VTAB 3 : HTAB 2
 30  PRINT "Please type your program's name."
 40  PRINT : INPUT ":  ";A$
 50  VTAB 8 : PRINT "Please type the number of one of the options: "
 60  PRINT : HTAB 5 : PRINT "1 - LOAD the program."
 70  PRINT : HTAB 5 : PRINT "2 - LOAD and RUN the program."
 80  PRINT : GET B$ : PRINT          : REM ** Must have a CR between GET and D$**
 90  IF B$ = "1" OR B$ = "2" GOTO 110
100  PRINT CHR$(7): GOTO 50
110  PRINT D$"OPEN HGR LOADER"
120  PRINT D$"WRITE HGR LOADER"
130  PRINT "POKE 104,64"
140  PRINT "POKE 103,1"
150  PRINT "POKE 16384,0"
160  PRINT "NEW"
170  PRINT "LOAD "A$
180  IF B$ = "2" THEN PRINT "RUN"
190  PRINT "PRINT"                  :REM ** Clear input **
200  PRINT D$"CLOSE"
210  HOME : PRINT CHR$(12)
220  VTAB 2 : HTAB 3 : PRINT "To begin, type: EXEC HGR LOADER"
230  END
```

NOTE : You should add a new first line to your own program as follows:

```
5 PRINT CHR$(4)"CLOSE HGR LOADER"
```

Listing 40.2 EXEC file maker for moving TXTTAB.

Using HGR2 to Make More Space for your Program

Once you've done the TXTTAB fiddle and can't avoid using the high res one screen in your program, then you've used up all the easy options for improving your space situation. If you don't need mixed text and graphics, however, you can change to the HGR2 memory layout (see Figure 40.4d). If you do this, you won't be wasting the lowest 6K of RAM as you did with the HGR TXTTAB setup.

The best way to use the HGR2 arrangement for large program texts is to let the program line space load normally without any POKEs to alter TXTTAB from its standard setting at $0801. However, you probably will want to use a LOMEM: statement to move the variable tables up above the top of the high res two display memory area. To do this, the first line in your program should issue the statement LOMEM: 24576. The result of using this arrangement is shown in Figure 40.4e.

Memory Expansion, Compression and Segmentation

There is a great arsenal of software weapons you can bring to bear on programs that still will not fit. However, before choosing your weapon and heading off to battle, you should give careful consideration to shelling out some bucks for some hardware to help you out of the mess. The software methods have the advantage of being cheap, but some of them have the serious disadvantage of being tedious, frustrating, time consuming and demanding of all sorts of ongoing attention.

When your program begins to approach the memory limits of your Apple, and you've already fiddled with LOMEM: and TXTTAB, you should try to make a brave decision about the future of your program. If it is nearly through growing, and you are certain there are no new subroutines, bells or whistles that need to be added, it may be worthwhile to go about squeezing out a few bytes here and there.

If you still have a lot you'd like to add, you are facing a choice between spending your programming time wrestling with memory requirements or spending your time wrestling with the real programming task you're supposed to be working on. If you spend the $350 to $500 for a 128K memory card, all you'll need to do is plug it into one of your slots and load a utility from disk and all your problems will be solved. For the II/II+, the two major contenders in this market are Titan Technologies (formerly Saturn), and Legend Industries.

Titan sells a utility called EMBER (Extended Memory Basic Interpreter) which lets you store up to four megabytes of variables and arrays on external memory cards. Your program line space itself must stay in the Apple's main memory, and still must dodge around the graphics screens, but you can use the full 24K between the top of high res one and the bottom of DOS. Further, since you can also move DOS out of main memory up into one of the RAM banks of the memory card, you can get another 10K of space in main RAM. The Ampercard program from Legend is similar, but requires some modest modification of your program. If you have ProDOS, you can use the additional RAM for expanding the program itself rather than the variables, through use of the powerful ProDOS CHAIN command (see below). This ProDOS

option is available automatically in all //cs and in //es with 64K in the auxiliary slot. You can also use it with other types of RAM cards if you have the proper disk emulation utilities, and, of course, you can CHAIN back and forth from your disk drive if you don't have extra RAM.

If you do get the proper hardware and utilities to ship your variables off to a RAM card, then you will no doubt find that working with 35K of program text, free use of hi-res one display memory, and essentially unlimited space for variables, arrays and strings represents an enormous improvement on the standard situation. And all with no programming effort on your part. If your needs are greater and your budget larger, you can also consider purchasing a 68000 co-processor board from Saybrook or ETC. The Saybrook is better engineered and more compact, but this is only important in very high performance applications. Both of these boards are sold with a completely rewritten Applesoft Interpreter that actually runs on the 68000 instead of on the Apple's 6502. These 68000 Interpreters will run your programs as currently written but you get effectively unlimited space for program text, variables, arrays and strings, and everything runs much much faster as well (see Chapter 41).

Program Compression

If your program is large enough to need compression, then any changes you make in the program to render it more space efficient will tend to cause numerous bugs to pop up. It may be easier to start from scratch than to do major surgery. Nonetheless, you can usually compress your program line space to about two thirds of its current size without changing any of the statements. This is done for you by the Applespeed program in the List Master package from Southwestern Data Systems. Applespeed chops, squeezes and rearranges your program into an incredibly dense form. You can save and run the compressed version, but there are some real frustrations that come along with using this system.

Applespeed (written by Roger Wagner) works by doing four things to your program text. In its first pass, it removes all the REM statements in your program. This is an extremely important feature to be aware of. One of the most destructive aspects of working with an oversize program is that you tend to skimp on REM statements and your program therefore becomes more and more difficult to read and understand for later debugging and maintenance. Applespeed lets you write in REMs to your hearts content and then create an uncommented version of your program for actual use.

The second pass of Applespeed shortens all of your variable names to just two letters. The Interpreter only uses the first two letters, and this feature encourages you to use full descriptive variable names as you write since they can all be tidied up later.

The third pass drastically alters your program by combining lines (i.e., `130 DIM X(20,15):HOME:TEXT:VTAB 12:A=2:` etc., all with just one line number). This is a very complex and tricky procedure since all GOTOs and GOSUBs must be altered to reflect the changes, and the work must be done carefully to avoid messing up flow of control during execution. The advantage of combining lines is that each line has five bytes of overhead no matter how many statements are on the line. When you combine lines, you add one byte for the colon, but save four bytes. The fourth and final pass renumbers the remaining program lines by ones to help shorten the line numbers in GOTO and GOSUB statements.

You can select how many of the four passes Applespeed should do. Even if you only do the first pass, you end up with two versions of your program: one documented with REMs and one shorter but without comments. As you continue to program and debug, you can make

short term additions in the compressed version, but it is important to copy all the changes onto the documented version and then generate a new compressed version.

If you use passes three and four, the situation gets much worse. Some of the combined lines are longer than 239 characters and so you cannot edit them to make changes. Further, since there are so many statements on a line, the usual error messages are much less helpful, and, once you do locate the problem, it's a bit of a bother to have to figure out where the change goes in the fully expanded version. You typically have to go back and make the change in the full sized version and then wait 10 to 15 minutes while a new compressed version is generated.

Of course, all of these problems result from Applespeed doing exactly what you want it to be doing. It's a wonderful program.

Saving Space in Variable Tables

If your problem is in your variable tables and strings, things are much more bleak. The densest way to store numbers is in an integer array. Integer and real simple variables take up the same amount of space, but a large integer array with the same number of data elements takes up only 40 percent as much space as a real array. In either case, when assigning values to arrays, remember to use the 0 elements in each dimension.

Beyond these modest steps, you can only get improvements by keeping careful track of where each variable is used, and then going about constantly reassigning the same variable name to different tasks in different parts of the program. Another Roger Wagner utility called Apple-DOC (Southwestern Data Systems) is helpful here because it can generate cross reference tables which will list all of your variables in alphabetical order and tell which line numbers each variable name appears in. This is extremely useful while you are first writing and debugging any program, and is indispensible if you are sufficiently foolhardy as to try to reassign variables to save space after a large program is written.

Breaking the Program into Pieces

The three remaining software options for dealing with oversized programs all involve breaking your program text into pieces. Splitting is used to cut your program text into two parts in order to make it fit around one of the high res screens. It doesn't help for extremely large programs, but if you can use it, you won't have to do any rewriting of your Applesoft program.

The other two options, chaining and overlaying, require very substantial advance planning, and are difficult to start using after a program is already written. Both of these methods let you write your program in segments, store the segments on disk, and then have the various segments automatically swap back and forth between the disk drive and main memory as the program runs. These two methods both permit virtually unlimited Applesoft program size so that if you do suspect size problems in advance, you may want to keep them in mind as you work. As a good rule of thumb, if you expect that a single spaced, 80 column printed listing of your program will be more than 15 pages in length, chaining or overlaying may be necessary.

Splitting Around a Hi-Res Screen

You approach splitting in much the same way as you approach the use of changes in LOMEM: and TXTTAB. First select the appropriate memory layout from Figure 40.2a based on which high resolution graphics screen or screens you are using, and then use the subroutine in Listing 40.1 to read the values of the program size parameters for your program.

Splitting is useful when there is extra space left over between the top of your array table and the bottom of your string storage space (see Figure 40.4b), or if you had put TXTTAB above high res screen one and now regret the waste of the first 6K of memory (see Figure 40.4c). As you can see from the splitting solutions shown in Figure 40.4h and i, this technique lets you tailor your program line space to fit precisely into the available memory space.

Another nice thing about this technique is that it is very simple to use. The actual splitting is done for you by a splitter program. All you have to do is to tell the splitter which area in Apple memory you want to use for graphics, and it takes care of the rest. There are several splitter programs available, some of which have bugs in them, but the splitter written by Charles Kluepfel is completely debugged and has been used successfully by thousands of Apple programmers since it was published in 1980.

You can purchase this program on disk from *Call A.P.P.L.E.*, or you can type it into your computer from Listing 40.3 and save it for free. The program is explained in detail in *Call A.P.P.L.E in Depth: All About Applesoft,* and a less technical explanation was written by Norman Kushnick in the January 1983 issue of the *Call A.P.P.L.E.* magazine. The version provided by *Call A.P.P.L.E.* assumes you have DOS 3.3 loaded in main memory, but the caption for Listing 40.3 explains how to modify the program for use if you have moved DOS into the high 16K.

To use the splitter program, you first load it into memory from disk, POKE in the information about your choice for graphics, load your Applesoft program, and then CALL the splitter (the exact instructions are noted in Listing 40.3). When the splitter is called, it begins scanning through your Applesoft program to find the last complete program line before your designated memory location for starting the split. It then moves the remainder of your program up above the graphics screen, leaving a long gap in the middle of your program. Finally, it goes about changing a large number of internal pointer and Applesoft parameters to make this gap invisible to the Applesoft Interpreter.

Once the splitter has finished its work, you can LIST or RUN your program and you will not notice any difference from its previous behavior except that your space problems will be eased. This splitter program is not completely effective, in that you cannot enter new program lines after the program has been split. To change your program, you reload it from disk in its intact form, make the changes, save it, and then load and call the splitter again. The most convenient way to use the splitter is to set it up with an EXEC file, and a program to make that EXEC file is shown in Listing 40.4.

Chaining Among Program Segments

One special attraction of chaining is that any beginning Applesoft programmer can use this technique perfectly well without paying any attention at all to the internal workings of the Applesoft Interpreter or to the program size parameters. Just use a little additional discipline while writing your Applesoft program so that it is made up of two or more separate parts. Within a program part, you can bounce around and use GOTOs and GOSUBs etc., but you must not use GOTOs and GOSUBs to cross from one program part to another program part.

Each program part is loaded into Apple memory from disk when it is needed, much as if it were a separate and distinct Applesoft program. What is special about chaining is that the various parts all share the same variable table, array table and string storage space which effectively remain in memory while the program text itself is being changed.

Listing 40.3 Splitter routine by Charles Kluepfel. You can get this routine on disk from Call A.P.P.L.E. user's group or you can enter it yourself, if you're careful.

Entering the program:

This machine language program is big enough to be a serious bother to enter directly, but too small to justify buying an assembler and learning to use it. To enter the program, you type: CALL-151 and then respond to the * prompt by typing 9400: The number is the location in Apple memory where the program will begin, and the colon tells the Monitor you are going to enter data. You then type only the Program Bytes. When you hit Return, you will be prompted with the next line number, but you must type the colon before proceeding. To check your work, type 9400L and you will see the disassembling listing, just as in this illustration. To save the program once it is entered, type: BSAVE SPLITTER, A$9400, L$10E. If you have an assembler, you can get the source code from Call A.P.P.L.E., "In Depth: All About Applesoft."

If you are using a DOS mover as well, the program should be placed at $B400. You only need to change two bytes to relocate it; these are the bytes marked in boldface in the figure at $94BA and $950C. Change these from 94 to B4, BSAVE the program as described above, but then do a BLOAD SPLITTER, A$B400, followed by BSAVE HISPLITTER,A$B400,L$10E.

Using the Splitter:

The program in Listing 4 will do everything for you, but for those who'd like to know: Your Applesoft program is first loaded from disk. Next, the hexadecimal value of the number of pages (256 bytes each) you want in the gap is POKEd into location $0006, and the page number where the break should begin is POKEd into location $0007. The move is then begun with a CALL 37888 (for the $9400 version) or with a CALL 46080 (for the $8400 version) if you've moved DOS.

Be sure you understand your space problem before you try the splitter. You must choose a break point which is within your program text (lower than PRGEND from Listing 1) and your program must not be so big that it will overwrite the splitter program (at 37888 or 46080) during the splitting process.

You cannot save or modify a program after it has been split with this splitter; it must be reloaded in its orginal form, modified, saved, and split again.

Splitter

```
Location
        Program         Dissasembled
          Bytes           Listing

**************************************************
***** Collect the user's parameters for the ************
*****     size and location of the gap          ************
**************************************************


9400 -  38              SEC
9401 -  A5 B0           LDA   $B0
9403 -  85 41           STA   $41
9405 -  65 06           ADC   $06
9407 -  85 43           STA   $43
9409 -  A5 AF           LDA   $AF
940B -  85 40           STA   $40
940D -  85 42           STA   $42
```

```
**************************************************
*****        Mark end of program          ***********
**************************************************

940F -  A0 06           LDY   #$06
9411 -  A9 00           LDA   #$00
9413 -  91 42           STA   ($42),Y
9415 -  88              DEY
9416 -  D0 F9           BNE   $9411

**************************************************
*****     Move the program segment        ***********
**************************************************

9418 -  B1 40           LDA   ($40),Y
941A -  C6 40           DEC   $40
941C -  91 42           STA   ($42),Y
941E -  C6 42           DEC   $42
9420 -  A5 42           LDA   $42
9422 -  C9 FF           CMP   #$FF
9424 -  D0 F2           BNE   $9418
9426 -  A5 41           LDA   $41
9428 -  C6 43           DEC   $43
942A -  C6 41           DEC   $41
942C -  C5 07           CMP   $07
942E -  10 E8           BPL   $9418

**************************************************
*****          Reset pointers             ***********
**************************************************

9430 -  A5 07           LDA   $07
9432 -  85 08           STA   $08
9434 -  C6 08           DEC   $08
9436 -  38              SEC
9437 -  A5 B0           LDA   $B0
9439 -  65 06           ADC   $06
943B -  85 B0           STA   $B0
943D -  85 6A           STA   $6A
943F -  85 6C           STA   $6C

**************************************************
*****  Scan through from beginning of     ***********
*****  program to find first line which   ***********
*****       extends into the gap          ***********
**************************************************

9441 -  A5 67           LDA   $67
9443 -  85 40           STA   $40
9445 -  A5 68           LDA   $68
9447 -  85 41           STA   $41
9449 -  A0 01           LDY   #$01
944B -  B1 40           LDA   ($40),Y
944D -  C5 08           CMP   $08
944F -  F0 0B           BEQ   $945C
```

```
9451 -  AA              TAX
9452 -  88              DEY
9453 -  B1 40           LDA   ($40),Y
9455 -  85 40           STA   $40
9457 -  86 41           STX   $41
9459 -  C8              INY
945A -  D0 EF           BNE   $944B
945C -  88              DEY
945D -  B1 40           LDA   ($40),Y
945F -  F0 02           BEQ   $9463
9461 -  D0 05           BNE   $9468
9463 -  C8              INY
9464 -  B1 40           LDA   ($40),Y
9466 -  D0 E9           BNE   $9451

*****************************************************
***** Set up an invisible GOTO for the gap ***********
*****  to force use of next line pointers  ***********
*****************************************************

9468 -  C8              INY
9469 -  C8              INY
946A -  C8              INY
946B -  C8              INY
946C -  B1 40           LDA   ($40),Y
946E -  D0 FB           BNE   $946B
9470 -  C8              INY
9471 -  C8              INY
9472 -  C8              INY
9473 -  C8              INY
9474 -  C8              INY
9475 -  A9 AB           LDA   #$AB
9477 -  91 40           STA   ($40),Y
9479 -  A5 40           LDA   $40
947B -  85 42           STA   $42
947D -  A5 41           LDA   $41
947F -  85 43           STA   $43
9481 -  C8              INY
9482 -  84 09           STY   $09
9484 -  A0 01           LDY   #$01
9486 -  B1 40           LDA   ($40),Y

*****************************************************
***** Set up 'next line pointer' to cross gap ********
***** and then correct all the pointers in ***********
*****         the moved segment          ***********
*****************************************************

9488 -  38              SEC
9489 -  65 06           ADC   $06
948B -  91 40           STA   ($40),Y
948D -  88              DEY
948E -  AA              TAX
948F -  B1 40           LDA   ($40),Y
9491 -  85 40           STA   $40
```

```
9493 -  86 41        STX   $41
9495 -  C8           INY
9496 -  A6 09        LDX   $09
9498 -  D0 11        BNE   $94AB
949A -  B1 40        LDA   ($40),Y
949C -  D0 EA        BNE   $9488

*****************************************************
*****  Table of numbers for calculating  ************
*****  ASCII rep. of decimal line number.  **********
*****  Monitor does incorrect disassembly ***********
*****************************************************

949E -  4C 00 00     JMP   $0000
94A1 -  10 27        BPL   $94CA
94A3 -  E8           INX
94A4 -  03           ???
94A5 -  64           ???
94A6 -  00           BRK
94A7 -  0A           ASL
94A8 -  00           BRK
94A9 -  01 00        ORA   ($00,X)

*****************************************************
*****  Generate and write the ASCII  ****************
*****  decimal line number in the GOTO  *************
*****************************************************

94AB -  C8           INY
94AC -  B1 40        LDA   ($40),Y
94AE -  85 19        STA   $19
94B0 -  C8           INY
94B1 -  B1 40        LDA   ($40),Y
94B3 -  85 1A        STA   $1A
94B5 -  A9 A1        LDA   #$A1
94B7 -  85 1B        STA   $1B
94B9 -  A9 94        LDA   #$94
94BB -  85 1C        STA   $1C
94BD -  A0 01        LDY   #$01
94BF -  A2 01        LDX   #$01
94C1 -  A9 2F        LDA   #$2F
94C3 -  85 1D        STA   $1D

94C5 -  E6 1D        INC   $1D
94C7 -  A5 19        LDA   $19
94C9 -  88           DEY
94CA -  38           SEC
94CB -  F1 1B        SBC   ($1B),Y
94CD -  85 19        STA   $19
94CF -  C8           INY
94D0 -  A5 1A        LDA   $1A
94D2 -  30 02        BMI   $94D6
94D4 -  A2 00        LDX   #$00
94D6 -  F1 1B        SBC   ($1B),Y
94D8 -  85 1A        STA   $1A
```

```
94DA - 10 E9        BPL   $94C5
94DC - 8A           TXA
94DD - D0 E6        BNE   $94C5
94DF - A5 19        LDA   $19
94E1 - 88           DEY
94E2 - 18           CLC
94E3 - 71 1B        ADC   ($1B),Y
94E5 - 85 19        STA   $19
94E7 - C8           INY
94E8 - A5 1A        LDA   $1A
94EA - 71 1B        ADC   ($1B),Y
94EC - 85 1A        STA   $1A
94EE - 84 1E        STY   $1E

94F0 - A5 1D        LDA   $1D
94F2 - A4 09        LDY   $09
94F4 - 91 42        STA   ($42),Y
94F6 - C8           INY
94F7 - A9 00        LDA   #$00
94F9 - 91 42        STA   ($42),Y
94FB - E6 09        INC   $09
94FD - A4 1E        LDY   $1E
94FF - C8           INY
9500 - C8           INY
9501 - 98           TYA
9502 - C9 0A        CMP   #$0A
9504 - 30 BB        BMI   $94C1
9506 - 86 09        STX   $09
9508 - A0 01        LDY   #$01
950A - 4C 9A 94     JMP   $949A
```

Listing 40.4 Automatic Splitter Manager. This program demonstrates the use of an EXEC file and error catching menus to make a "user friendly front end" that handles the tricky points of initializing your program. Lines 410 to 480 do most of the work, but the full program makes life a lot easier.

Once the EXEC file (PROGRAM CUTTER) is created, it can be used repeatedly without use of the manager. To make this a true turnkey system, you can put a new HELLO file on the disk. Type NEW, then:

```
10 PRINT "Please wait a few momments"
20 PRINT CHR$ (4) "EXEC PROGRAM CUTTER"
```

Now, type: SAVE HELLO. In the future, when the disk is booted, the EXEC file will run automatically, it will load all the necessary files, do the POKEs, split your program and then RUN it.

Automatic Splitter Manager

```
 10  D$ = CHR$(4): HOME: PRINT CHR$(12)
 20  VTAB 3 : HTAB 5 : PRINT "This Program Creates an Exec File called:
 30  PRINT : HTAB 15 : PRINT "PROGRAM CUTTER"
 40  PRINT : PRINT: HTAB 5 : PRINT " When you call it with  EXEC PROGRAM CUTTER"
 50  HTAB 7 : PRINT " it will load your program, split it, and run it"
 60  PRINT : PRINT : HTAB 5: PRINT " Before you use this, you must add a new"
 70  HTAB 7 : PRINT "first line to your program to close the Exec File."
 80  HTAB 7 : PRINT "That new line is: "
 90  PRINT: PRINT : HTAB 7 : PRINT " 5  PRINT CHR$(4)"CHR$(34)"CLOSE PROGRAM
       CUTTER"CHR$(34)
100  PRINT: PRINT "HIT ANY KEY TO CONTINUE": GET K$
110  HOME: PRINT CHR$(12)
120  VTAB 2: HTAB 3 : PRINT "Please type your program's name, then hit 'RETURN'"
130  INPUT "";A$
140  PRINT: PRINT "Is this correct ? Y/N"
150  INPUT "";B$
160  IF B$ = Y OR B$ = "y" GOTO 180
170  PRINT CHR$(7): GOTO 110
180  PRINT: PRINT "Please type the number of one of the options, then  hit 'RETURN'"
190  PRINT: PRINT: HTAB 7 : PRINT " 1 - I have SPLITTER on this disk"
200  PRINT: HTAB 7: PRINT "2 - I have moved DOS & have HISPLITTER on this disk"
210  VTAB 17: INPUT "";C$
220  IF C$ = "1" OR C$ = "2" GOTO 240
230  PRINT CHR$(7): GOTO 210
240  HOME :PRINT CHR$(12)
250  VTAB 3: PRINT "Please type the number of one of the options, then
       hit 'RETURN'"
260  PRINT : PRINT: HTAB 7 : PRINT "1 - Split around Hi Res Screen 1"
270  PRINT : HTAB 7 : PRINT "2 - Split around Hi Res Screen 2"
280  PRINT : HTAB 7 : PRINT "3 - Split around Hi Res Screen 1 & 2"
290  PRINT : HTAB 7 : PRINT "4 - Split around  Hi Res 2 with an extra 2K left
       free above $6000"
300  VTAB 15 : INPUT "";E$
310  IF E$ = "1" OR E$ = "2" OR E$ ="3" OR E$ = "4" GOTO 330
320  PRINT CHR$(7): GOTO 300
330  E = VAL(E$)
340  ON E GOTO 350, 360, 370, 380
350  L = 32 : B = 32 : GOTO 390
360  L = 32 : B = 64 : GOTO 390
```

```
370   L = 64 : B = 32 : GOTO 390
380   L = 40 : B = 64
390   IF C$ = "1" THEN P$ = "SPLITTER" : C = 37888
400   IF C$ = "2" THEN P$ = "HISPLITTER" : C = 46 080
410   PRINT CHR$(4)"OPEN PROGRAM CUTTER"
420   PRINT CHR$(4)"WRITE PROGRAM CUTTER"
430   PRINT "LOAD "A$
440   PRINT "BLOAD "P$
450   PRINT "POKE 6 "L: PRINT "POKE 7,"B
460   PRINT "CALL "C
470   PRINT "RUN"
480   PRINT CHR$(4)"CLOSE PROGRAM CUTTER":HOME : PRINT "Do the EXEC now "
```

In a typical arrangement, part A might collect data from the keyboard and store the data in variables. On completion it would cause a chain to some data analysis routines in part B stored on disk. Part B would be loaded and completely overwrite the program line space of part A, but all the data stored in variables would still be available. After analysis there might be a chain to part C for doing graphics. At any point, the user could select from a menu an option to return to the data entry part, etc.

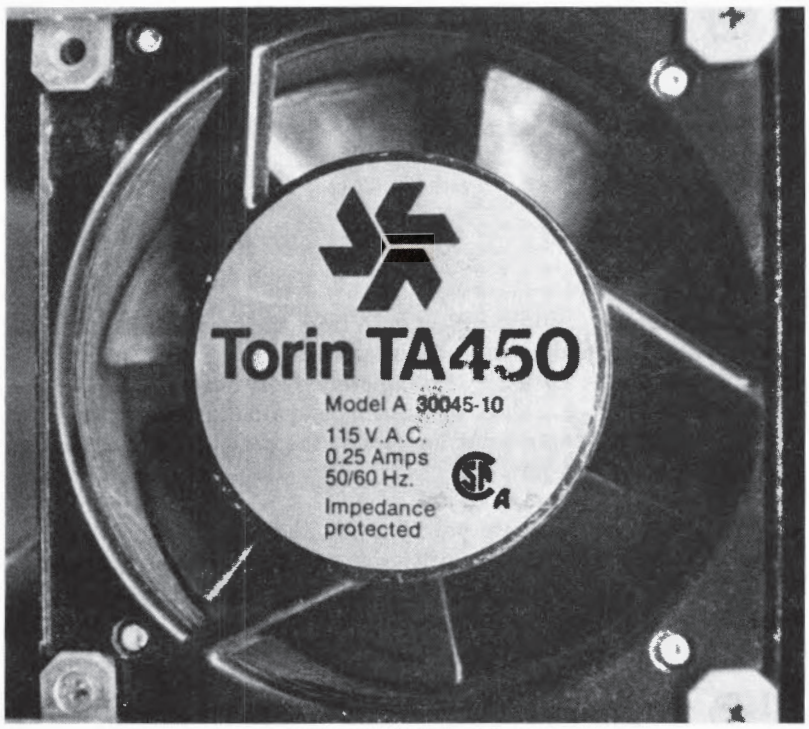
To chain in Applesoft, you use the chain program provided with your DOS 3.3 master disk and follow the instructions laid out in your Applesoft manual. If you have ProDOS, it's much simpler. There is a CHAIN command which you issue like any other ProDOS command (i.e., OPEN, CATALOG, etc.). Further, you can specify the line number in the new part at which program execution should begin after the load is complete.

The ProDOS CHAIN command is the key to using large amounts of extra RAM memory for large Applesoft programs. The extra RAM is configured as a RAM disk drive emulator and your program segments are stored as files on the RAM drive. ProDOS will automatically configure 64K in the //e or //c auxiliary slot as a disk emulator, but any RAM disk with appropriate ProDOS disk emulator software will do. You then use the ProDOS CHAIN command with a line number specification much as you would ordinarily use a GOTO command.

Overlays for Increased Performance

An overlay is similar in nature to a Chain, but it is much more difficult to use. As with chaining, program segments are stored on disk and then swapped into memory; however, you usually leave one master program segment in memory at all times and swap only a smaller part of the program. One advantage of overlaying is that the changes can take place more quickly since you tend to load smaller segments than you would use with chaining, but this is not too important when you're working with a RAM disk. Further, if you need to have a common core portion of your program loaded at all times, chaining would require you to store a copy of that core in each program part kept on your disk thus using up more disk space.

Unfortunately, overlaying is not directly supported by DOS 3.3 or ProDOS, and the utilities that are available for carrying it out are not extremely well polished. There is a perfectly adequate overlay program in David Lingwood's *Amplifying Applesoft* article in *Call A.P.P.L.E. In Depth: All About Applesoft,* but it's a bit clumsy to use while you're still debugging or changing your core program.


Torin TA450
Model A 30045-10
115 V.A.C.
0.25 Amps
50/60 Hz.
Impedance
protected

Chapter 41

Speeding Up Applesoft

Speeding Up Applesoft Programs

While an Applesoft BASIC program is running, the Interpreter spends its time in a repetitive execution cycle which is described in detail in Chapter 38, and reviewed briefly below. If your program is not running fast enough, you may be able to trace the problem to one of five major events during the execution cycle. Some programs can be sped up by a factor of five or 10 times just by making a few adjustments in your commands or adding a few PEEKs and POKEs. Others require modification by a speed up utility such as Applespeed, or a compiler. Still others are immune to any software based improvement and require the purchase of speed up hardware.

It is worth noting at this point, before plunging into a meticulous study of your problem, that if the program is for your own use on just your own Apple, then there is a simple, buckshot, cure-all approach to solving all Applesoft speed problems without any particular analytic thought or programming effort on your part. Just buy a coprocessor board that runs faster than the Apple's native 6502 but still understands Applesoft.

The two options in this realm are to get a board with a new high speed 6502C (Saturn Accelerator, MCT Speedemon), or to buy an ETC or Saybrook 68000 coprocessor with a new 68000 based Applesoft Interpreter. The 6502C boards speed up everything by a factor of 3.6, and you can choose a 68000 board to get anywhere from five to 15 times the speed. Your minimal cost is about $300 for the MCT board, and your maximum cost is $2,000 for the fastest version of the Saybrook 68000 card.

These prices may be sobering, but the simplicity and finality of this solution makes for an eloquent argument in favor of shelling out the bucks. Some programs may require slight modification to work on these boards, but the yield in speed improvement versus programming effort will be far greater with the boards than without them.

Time for Five Events in Applesoft Execution

There are five potential bottlenecks which cause trouble for most programs. Each of these fits into the larger scheme of program execution which is stepped through in detail in Chapter 38, but they can be described here in simple, if less accurate, terms. The first three problems are fairly easy to do something about and result mostly from minor inefficiencies in the operation of the Applesoft Interpreter. The first has to do with the way GOTO and GOSUB

commands are carried out, the second is to do with getting variable names matched up with their assigned values, and the third turns up when you do a lot of reassignment of string variables.

The remaining two problems are much tougher nuts to crack because the Interpreter actually does a very good job, but is betrayed by the limitations of the hardware itself. These two bottlenecks are in the execution of the floating point math routines and the high resolution graphics plotting routines. The parts of the Applesoft Interpreter which do the floating point math and the plotting calculations are extremely efficient machine language subroutines, so there is no software fix. You have to buy supplementary hardware to improve calculation speed.

Speed Considerations in Locating Program Lines

For every line number in your program, there is a Program Line Field (PLF) stored in memory which contains the line number and a coded version of the program statement or statements on that line. When the program is run, the Interpreter must first locate the PLF it is supposed to be working on and then go about translating your program statements into short little machine language programs the 6502 can execute.

Locating a PLF is very easy when there are no branches or loops. The Interpreter can just step straight along through your program. When it comes to the end of one PLF, it can just skip over exactly five bytes and it knows it will find the beginning of the next PLF in your program sequence (see Chapter 38, Figures 38.1 and 38.2). However, things get a little more tricky when there is a branch or a loop.

The Interpreter is fairly efficient with FOR/NEXT loops. When it encounters a FOR statement, it makes a note of the actual address in Apple memory where the FOR loop begins. When the NEXT turns up it can just jump back to the proper address in memory, and there it will find the first PLF in the loop again and again until the FOR condition is met and execution escapes from the loop.

Things are not so tidy when the loop is caused by a GOTO or a GOSUB. The Applesoft Interpreter has no means of knowing the address in Apple memory where it can find the PLF bearing the line number specified in your GOTO or GOSUB. From the point of view of the execution section of the Interpreter, the PLFs of your program are stacked one on top of the next with no particular regularity except that they are in sequence. The line number stored at the beginning of each PLF has no more meaning than a name such as Joe or Martha.

To execute a GOTO or a GOSUB, the Interpreter will usually go all the way back down to the beginning of your program and step all the way back up through your program. Along the way, it pauses at the beginning of each PLF, compares the line number there with the one it is looking for, and then jumps to the beginning of the next PLF, and so on until it finds what it is looking for. If, for instance, the five hundredth and five hundred and first lines of your program read as follows:

```
5000 PRINT X= X + 1
5010 IF X < 24 GOTO 5000
```

then, the Interpreter will scan through all of the preceeding 500 PLFs before each execution of the two statements. Before this tiny, trivial loop is done, it will have had to scan 12,000 PLFs. Needless to say, this is not good for speed of program execution.

Actually, the Interpreter is just a little bit more careful than has been suggested. Before going all the way back to the beginning, it subtracts the current line number from the line number mentioned in the GOTO or GOSUB. If the result is greater than 256, it scans forward from the current position rather than going back to the beginning. This means that:

```
5020 GOSUB 5276
```

will execute fairly crisply.

Solving the GOTO/GOSUB Problem by Hiding Lines

If you must use a GOTO or GOSUB over a short distance, you can force the Interpreter to be more efficient with a few PEEKs and POKEs. The objective here is to fool the Interpreter by hiding the beginning of the program. You temporarily modify the internal working parameters of the Interpreter so that it believes the program begins just before the start of your loop. This avoids the long scan before each pass. On exit from the loop, you carefully restore the parameters to their original condition and then let the Interpreter go on about its work.

The fix for this inefficiency was written long ago by Roger Wagner, and is done as follows. At the very beginning of your program, insert the following line:

```
1  P1 = PEEK (103) : P2 = PEEK (104)
```

The Interpreter stores the true address of the first PLF in your program in locations 103 and 104 (TXTTAB—see Chapter 40), and this line simply copies these correct values into the variables P1 and P2 for later reference.

Now, just before the offending loop (in our example, just before line 5000), insert the following line:

```
4099  PX = PEEK (121) + 256 * PEEK (122) + 1:
      POKE 103, PX - INT (PX/256) * 256:
      POKE 104, INT (PX/256)
```

As the program runs, the Interpreter always stores, in locations 121 and 122 (OLDTEXT), the Apple memory address of the end of the PLF it is working on. This inserted PEEK and POKE statement copies that address out of its storage location in 121 and 122 (OLDTEXT) into variable PX, does some necessary math, and then shoves it into 103 and 104 (TXTTAB). Once this line has been executed, the previous five hundred lines of our example program have simply disappeared. The GOTO statement in our line 5010 now executes hundreds of times more quickly.

To return everything to normal, you need to add one more line just after the loop as follows:

```
5Ø11  POKE 1Ø3,P1 : POKE 1Ø4,P2
```

This just restores the original value of TXTTAB which we had cleverly squirreled away in P1 and P2 at the outset.

Using Applespeed to Improve GOTO/GOSUB Performance

A very different alternative to this PEEK and POKE approach is to use the Applespeed utility included with the List Master package from Southwestern Data Systems. The features of Applespeed are discussed in Chapter 40, but just one of its actions is important here.

Applespeed does not actually change the contents of your program, it just sort of rearranges it. The combine lines option scans through your program and puts as many statements as possible into a single PLF. A program with five hundred line numbers can often be rearranged into one with just 80 or 90 line numbers. The advantage of this is that when a GOTO or GOSUB causes a search for a line number, there will always be much less work for the Interpreter to do.

Many of the combined lines are quite long, but the length of a line has no effect on speed of execution of GOTOs and GOSUBs. The Interpreter is able to skip from the beginning of one PLF to the beginning of the next without scanning through the entire PLF. Therefore, it is the number of PLFs rather than the length of the program which causes GOTOs and GOSUBs to execute slowly.

Using Applespeed has the advantage of being a simple way of automatically speeding up all of the GOSUBs and GOTOs in your program as well as compressing your program into a smaller space. The disadvantage is that the compressed version is difficult to read and debug.

From BASIC Statements into Machine Language Routines

Once the Interpreter has found its way to the Program Line Field it is supposed to act on, it goes about setting up a series of machine language subroutines to carry out your program statements. Contrary to popular belief, this translation process is actually extremely fast. Each Applesoft command is represented in memory by a one byte token (see Chapter 38, Table 38.1). These tokens are not arbitrary, rather they refer to positions in the command table (see Chapter 38), which is the heart of Applesoft.

The command table is a neat list of the addresses at which each of the Applesoft execution routines begins. There is one such routine for each Applesoft command. The value of the token for a given Applesoft command tells exactly where the entry point to its execution subroutine is stored in the command table. The actual conversion from command token into an executing subroutine takes about 25 millionths of a second.

Laying Out Variables for Fast Location

However, now that you've got an executing machine language subroutine, things can slow down again because of the way Applesoft handles variables. Just as with line numbers and GOTO statements, Applesoft has trouble here because it has no quick way of finding out exactly where to look in Apple memory to find the stored value assigned to that variable.

For each simple variable in your program, the Interpreter sets up a Simple Variable Field (SVF; see Chapter 39) which contains the first two characters of the variable's name, and the value currently assigned to the variable. These SVFs are stacked one on top of each other in a variable table. As your program begins to run, the Interpreter sets up an SVF for each variable it encounters, in the order in which it encounters them. This means that the first pass through your program will be slowed by the necessity of creating all those SVFs, and that locating any given SVF when it is referred to again later on in your program will not be an extremely efficient process.

To access an SVF later on, the Interpreter starts at the bottom of the variable table and steps through, SVF by SVF, comparing the stored variable name in each one to the name it is looking for. When your program says 530 PRINT X2, the machine language PRINT routine leaps into life almost instantaneously, but then it asks the Interpreter to find out the current value of X2 before proceeding. If X2 happened to be the 83rd variable mentioned in your program, then a long search will ensue before execution of the PRINT routine can continue.

The easiest way to improve the performance of Applesoft in this regard is to give careful thought to the order in which variable names will be first encountered by the Interpreter during execution. If there is a loop in your program that executes 1000 times and uses the variable X5, then you will be richly rewarded if you add the following new first line to your program: 1 X 5 = 0. This will ensure that the SVF for X5 is the first SVF in the Variable Table and so will be found quickly whenever it is referenced.

The XREF utility in the Apple-DOC package from Southwestern Data Systems will examine your program and give you a list of all the variables. Next to each variable name, it lists all the line numbers on which you mention that variable. You can examine the occurrences and decide which ones get called frequently in loops. Armed with this information, you can select the most heavily used variables for early mention, thus making them easy to find in the variable table.

Array variables are organized into Array Variable Fields (AVFs) in a separate array table (see Chapter 39) and most of the same considerations apply. Once an AVF has been located in the array table, the position of any element within the array is rapidly calculated based on structural data stored just after the variable name. Therefore, from the point of view of execution speed, it doesn't matter which element in the array is being referred to.

Constants, Integers and Real Numbers

Some of you may be thinking that you could avoid the whole search process by specifying numbers as constants in the program text rather than as variables. Unfortunately, this is usually a very poor choice because numbers in the program text require considerable manipulation before they can be passed on to the execution routines. All of the Applesoft math routines work only with floating point numbers in a strictly specified format (see Chapter 39, Figure 39.2). Numbers in the program text are stored as ASCII representations of the digits.

The ASCII codes must first be converted to digits, next the digits are multiplied and summed to produce an actual number, and then the number is converted to its floating point representation. In nearly all cases this is slower than a search for an SVF.

This requirement for a floating point format also makes the use of integer variables unwise. When an integer variable is referenced, it is first converted into its floating point representation and then acted on. This conversion is repeated with every reference. Real variables are stored in their full floating point representation. Integers save no space in memory when they are in simple variables, although they may be justified for setting up more compact arrays.

Special Considerations for Strings

The method that Applesoft uses for storing strings is fast and space efficient, however, programs which do a lot of reassignment of string variables can trigger a serious speed problem. The SVFs and AVFs which describe the strings are fairly well behaved, but the strings themselves are stored separately in memory in a string storage space (see Chapter 39, Figure 39.3) which is not so well behaved. The problem is that when a new string is assigned to a string variable name, the old string is left abandoned in memory. The old abandoned strings accumulate, taking up more and more space as the program runs.

This process continues until there is no more free space left in memory, at which time program execution grinds to an abrupt halt while the Interpreter cleans house (see Chapter 40, Figure 40.1f and 1g). This process, called garbage collection, involves the compaction of all the active strings into a small neat bundle. The algorithm that Applesoft uses for the compaction process is unfortunately very inefficient.

The two solutions to this problem are to force frequent compaction before the strings get too disorganized or to replace the algorithm with a faster one. By placing the statement:

```
X = FRE (0)
```

at various points around your program you can cause a large number of faster compactions. This is a bit of a drag on overall execution speed but avoids any gross halts and pauses.

If you have ProDOS, then the problem is much less severe because ProDOS acts to patch Applesoft with a dramatically enhanced compaction algorithm. Otherwise, you can get fast garbage collection routines from such commercial sources as the Ampersoft package from MicroSparc which speeds up the process by about 300 times.

Compilers

Three of the most time sensitive aspects of program execution can be sharply enhanced by using a Compiler to modify your program before it is run. The Applesoft Interpreter's greatest problems arise from the fact that it does not have absolute addresses for finding line numbers used in GOTOs and GOSUBs or for finding variable fields. In both cases, it has to do a long series of search and compare loops before execution can proceed. The third problem is that it cannot do integer math, but must do all its math with 40 bit floating point equivalents even if you just want to add 1 + 1.

Compilation can improve the running speed of some programs by two to five times. However, there are some disadvantages: compilation can double the size of a program and it is extremely difficult to debug a compiled program. The code expansion is caused because the Compiler does much of the work of setting up the variable fields and tables before the program is run.

The Einstein Expediter II produces fairly fast, compact code but the trade off for this good result is that it takes a long time to do its work. Ideally, you write and debug a program using the standard Applesoft Interpreter, and then compile the program only after it is finished. If you still have debugging to do however, then the long waits for each recompilation are a little frustrating.

Speedstar from Southwestern Data Systems plays the other end of the market by doing the compilation rapidly but producing code which is large in size and does not run as fast. The TASC Compiler from Microsoft has the advantage of being well known by many programmers but doesn't excel in either compilation speed or execution speed. A review of these three compilers and the Hayden Applesoft Compiler appears in *The Book of Apple Software 1984* by Stanton, et al. (The Book Company).

Hardware for High Speed Math Routines

If you've gone through and tidied up your program to make the most efficient use of the Applesoft Interpreter or have gone ahead and started using a compiler, then you've exhausted the software options for speeding up program execution. As explained earlier your best option at this point is to buy a fast 6502C based coprocessor board or a 68000 board.

However, in a similar price range, there are a few other hardware options which may be even more helpful for some programs. These are cards which specialize in speeding up the floating point math routines or the high resolution graphics plotting routines.

FTL for the 8088

The most generally useful of these cards is the ALF AD8088 coprocessor board. This card does not run your entire program on the 8088, rather it intervenes whenever Applesoft tries to run one of its floating point math routines. The entire Applesoft Interpreter is copied into the high 16K of Apple RAM before your program starts to run (this won't work on a 48K Apple), and the FTL program then goes in and modifies the math routine entry points in the copy of the Interpreter so that it will hand over control to the 8088 instead of doing the math itself.

An added advantage of this scheme derives from the fact that the various Applesoft compilers also try to use the built in floating point routines in the Applesoft Interpreter. The FTL program is able to send math work out to the 8088 even when running with a compiled version of an Applesoft program.

The ALF AD8088 can also run CP/M 86 and MS-DOS, and ALF makes the DC3 disk controller which can read and write IBM PC compatible disks. Therefore, you get a lot more than just fast math for your money. There is an option to add an 8087 floating point math coprocessor (see Chapter 30), but the overhead involved in passing numbers from the 6502 to the 8088 and then to the 8087 tends to sharply restrict the advantages of using an 8087. The major MS-DOS software firms have chosen to support the Rana 8086/2 (see Chapter 30) rather than

the ALF AD8088 so this is not the best bet if you want IBM PC software compatibility, however if you have a large amount of programming time invested in an Applesoft basic program that has math speed troubles, the AD8088 is a good card to keep in mind.

The AMD 9511 Floating Point Processor

One final option is the Model 7611 Arithmetic Processor from California Computer Systems. This card is based on a two MHz version of the AMD 9511 floating point processor. Just as with FTL, your program runs on a modified version of the Applesoft Interpreter and most math gets farmed out to the math processor. For tasks like multiplication and division, this doesn't do all that much for you. However, for complex functions such as the calculation of hyperbolic cosines or just doing exponentiation, speed can increase by a factor of ten, and the manufacturers claim an improvement of seventy fold for square roots. In addition, this system takes advantage of the little used USR () command in Applesoft (see Chapter 38) to add a variety of math commands not normally available in Applesoft.

The FORTH Option on the 9511

The fastest way to run any Applesoft program is on the 12.5 megahertz version of the Saybrook 68000 coprocessor which costs $1200. However, if you're planning a new project which uses a huge amount of math, your best choice may to switch to FORTH instead of BASIC and purchase the MicroSpeed system from Applied Analytics. That system takes advantages of the inherently fast execution algorithms in FORTH and then boosts them with a four MHz version of the AMD 9511. In addition, Applied Analytics sells a 4MHz 6502C board (actually the same board that Saturn sells as The Accelerator), which boosts the general speed of the FORTH program.

With the 6502C board and the fast AMD 9511, math intensive FORTH programs can run up to 100 times faster than their Applesoft equivalents. This is much better than you can do with a 68000 running Applesoft, and there are astronomy and engineering applications available through a user's group. This system has been used in a telescope driver system for the space shuttle.

Address Translation for Hi-Res Plotting

A second speed bottleneck which cannot be improved by compilers or other software fixes is the Apple's high resolution graphics plotting system. In theory, the math involved in figuring out where points should be placed on the screen is not extremely complicated. The problem is that the Apple's display memory has a rather odd and complex internal organization.

The source of the problem is deep in the heart of the Apple's Video Display Generator (VDG) system. The VDG runs separately from the 6502 at very high speed (14 megahertz), and is unable to perform the calculations necessary to create a linear mapping of locations in memory to positions on the screen. The 6502 is forced to do all the work to figure out where to put things to unscramble the VDG's odd scanning pattern (see Chapter 21, Figure 21.7).

Originally, this limitation derived from the technology available to Wozniak in 1976 when the Apple was designed. However, the odd mapping pattern was subsequently written into all the graphics software that runs on the Apple. When the //e and //c were designed, the addresses were left scrambled to ensure compatibility of software between the II+ and //e.

Two companies have developed unscrambling hardware which lets you plot points as if the video addresses were linear. This leads not only to speed increases, but to much greater simplicity of programs for three dimensional graphics, rotations, etc. The unscrambling is done by Programmed Array Logic (PAL) chips rather than by microprocessors. A PAL is a hardwired array of logic gates which perform the same function every time they are used. Dump a video address in one side, and the mapped version pops out the other side of the PAL almost instantaneously (two millionths of a second to be exact).

The problem with these systems is that they were not developed until 1984, by which time, coprocessors such as the Accelerator and the Saybrook board had greatly alleviated the problem by sheer brute computational force. The ADGS system from ALF goes with their 8088 board (see above) and has the added advantage of handling super hi res on the //e. A less expensive stand alone product from Magellan Computer called Speed-Grafix was announced in 1983, but interested users will have to contact Magellan to confirm availability.

Super Hi-Res Plotting for the //e and the //c

The ability to plot in double resolution on a //e or a //c with an extra 64K in the auxiliary slot makes the mapping problem even worse. Fortunately, the new 68000 based Applesoft Interpreter from ETC includes super hi-res plotting routines. You just issue an HGR2E command and go about plotting in a 560 × 192 dot space. This feature alone makes the ETC 68000 board a good choice for anyone considering a large investment ($1,450) for a major upgrade of their //e. This board will also solve a variety of other program space and program speed problems.

The somewhat less expensive ADGS system from ALF also helps with //e super high res plotting as well as providing the basis for an MS-DOS 8088 system. However, if you need super high res plotting, but aren't overly concerned about speed, you can get the HGR6 plotting routine package from ALF which runs directly on your //c's 65C02 or the //e's 6502. These routines give you full control over all 16 colors as well as over the 560 × 192 dots.

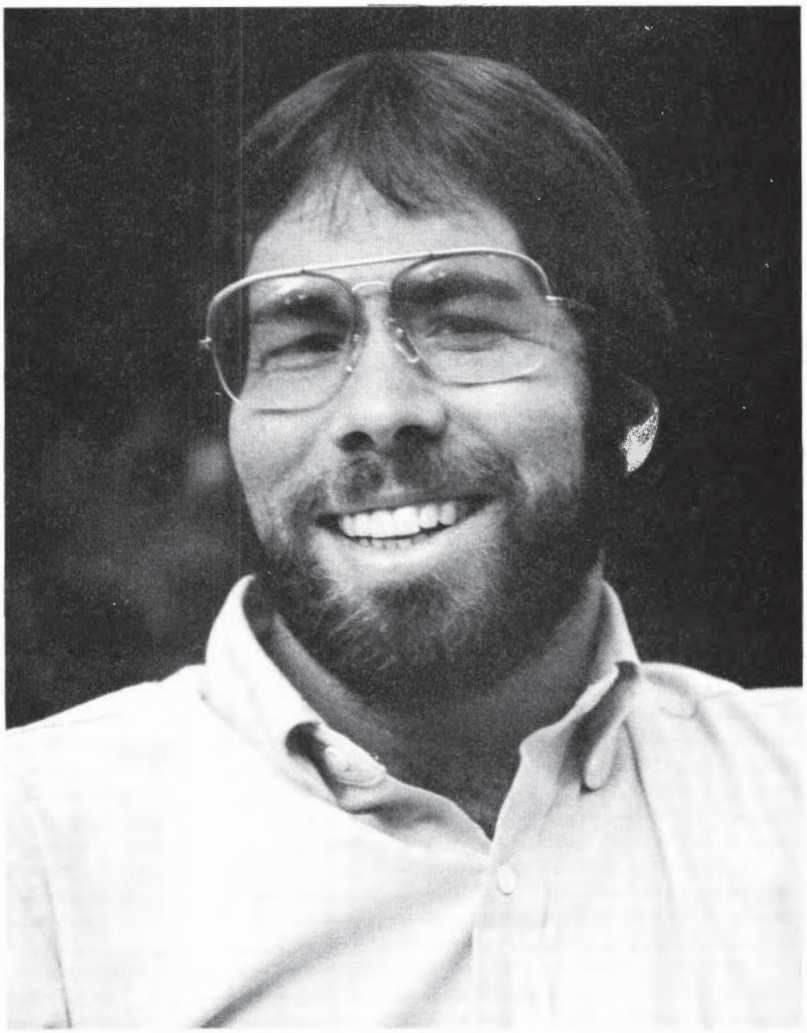
High Speed Vector Math for Ultra Hi-Res

If you're willing to do some rewriting of your program and are willing to spend $1,000 for truly spectacular high speed graphics, you should consider buying the Number Nine Graphics System. This board is based on the NEC 7220 graphic display controller. You can use ampersand commands from within your Applesoft program to plot in 640 × 400 dot resolution with 16 colors. A more expensive version gets you 1000 × 1000 resolution and things get better with each $1000 you're willing to throw in.

The NNGS board comes with 128K of display memory, and the NEC 7220 chip that manages the display memory can do truly mind boggling things on the screen. This chip is actually a microprocessor, but instead of commands like "load accumulator" or "add with carry," it has machine language instructions like "draw circle" and "fill rectangle." This board can do animation tricks in a second which might take five minutes on an unadorned Apple //e or IBM PC. (It's worth mentioning that the engineers at Number Nine Computer Systems who designed the NEC 7220 board are the same people who actually designed the 4 MHz 6502C board sold by Saturn as The Accelerator and by Applied Analytics as their math booster).

IV

COMMENTED APPLE



Chapter 42

Steve Wozniak: Designer of the Apple II

Steve Wozniak lives in a wonderful wooden castle, nestled high in the mountaintops above California's famed "Silicon Valley." The hall outside his home lab is lined with video arcade machines, and inside counters are covered with the expected armada of Apple II+s mixed in with video equipment. In another room, bookshelves are stuffed with hundreds of software packages. Steve said he regularly makes the rounds of the local computer stores to see what's being sold and buys up whatever looks interesting. Like the rest of us, he has a few favorite new peripheral cards from various manufacturers, as well as one or two that he can't seem to get working.

Although Wozniak designed the original Apple II/II+, and wrote much of the Monitor and DOS 3.3 system software, he later stopped working on design at Apple. While away, he worked on an uncompleted engineering degree, among other things. He said he feels the company had neglected the Apple II during years of advertising emphasis on the Apple III and development work on MacIntosh and the Lisa, but that all that was changing now.

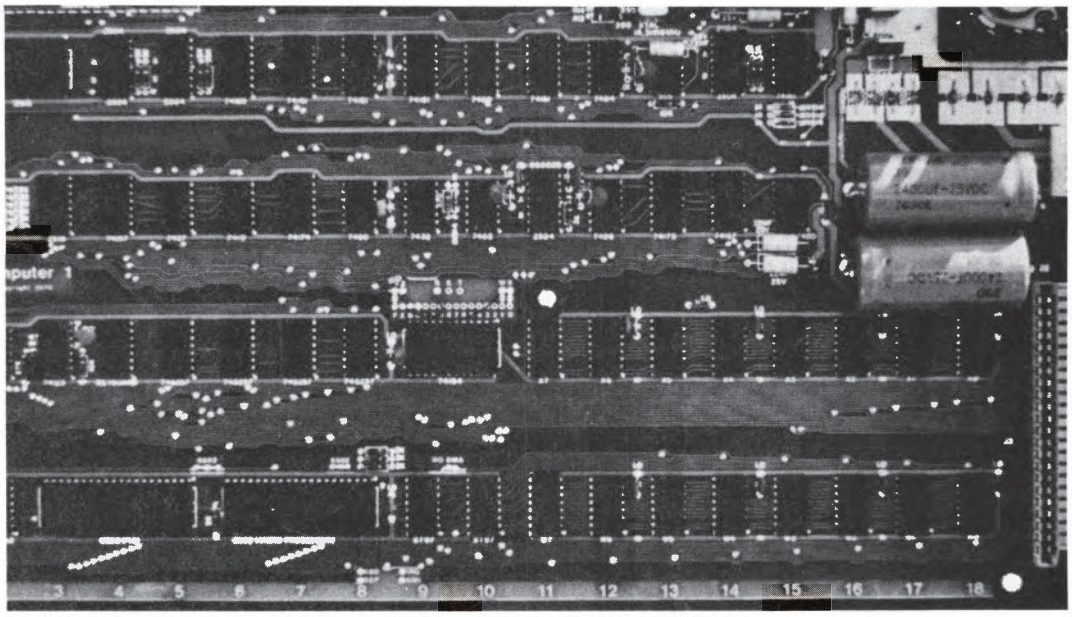


Fig. 42.1 Apple I motherboard.

During Wozniak's absence, Walter Broedner had come from Synertek, convinced that he could pack most of the circuitry of the Apple II into one or two custom circuit chips. When Broedner produced a successful design, Apple decided to go with it because it would greatly simplify the task of assembling the Apple II. Thus the //e was born largely through Broedner's individual efforts. Sales of the //e soared, John Sculley came to Apple as president and decided to make a strong commitment to the Apple II line, the //c design was launched, and Wozniak was drawn back in to head design work on future Apple II products.

At the time of our visit in late November of 1983, the //c was already built, so the most exciting thing on the horizon was the new 65802/65816 chip which had just been announced publicly a few days earlier at COMDEX. This chip changes the whole ballgame for the Apple II family. Wozniak had been very enthusiastic about ProDOS when we talked two months earlier, but now it seemed an even newer operating system would be needed to manage large amounts of memory and to take advantage of the speed and power of the new chips.

Interview

The following formal interview was conducted over the telephone by Aaron Filler, from Cambridge, MA., in September of 1983:

Aaron Filler: What do you do on a day to day basis?

Steve Wozniak: In theory, I design computers. In reality, I spend about 50 percent of my time answering phone calls, doing a lot of interviews. I travel pretty regularly to computer user clubs and I speak. I've been doing that for about four years. I've got virtually nothing going on outside of Apple at the moment, nothing serious.

Filler: What do you think a personal computer should be able to do ideally in the future, other than the kinds of things people use them for now?

Wozniak: Well, it ought to have such incredibly sophisticated software that it can teach you in a very entertaining way, and I don't believe that's possible today.

Filler: Do you think as time goes by people will program less and learn less about computers or will there be more programming and more interest in what's inside of them?

Wozniak: That's funny, because we're in the dilemma right now where we've pretty much envisioned the world as just wanting solutions, only solutions. No one wants to have to learn how to program, they just want to buy a canned program. And at the same time, we're starting to educate more and more people as to how to program. I think there will always be more programmers, and eventually, when you look to purchase a computer, there will be a tremendous market for people who know how to program and want a great tool that they can even use with their own programs.

The most minimum thing of all is that they (computer users) buy a spreadsheet. Everyone knows a spreadsheet is no longer simply just a formula. A formula is a minimum, a program is a maximum. But I think there's a lot of room for. . .. I think the right language is not here yet.

Of course, a lot of incredibly great, formal, theoretical work has gone into developing some of the languages in existence. But the reason I say that no language has been developed, basically,

since the advent of microcomputers, and since their applications and their user interface standards became pretty acceptable, is that there has been no new language based on those type of applications and user interfaces.

I think the first steps involve basic support. The languages in existence, BASIC and Pascal, started changing to include units, procedures and commands to do things like graphics and color graphics, which became standard on personal computers. But there was no new language based around knowing that it's got a certain size machine, approximately so much memory and that it's going to be doing these sorts of things. And I think that there's a lot of headway to be made there.

Filler: On the Apple I, wasn't there a PIA (Peripheral Interface Adapter) on the board?

Wozniak: Yes, there was. It was connected to a video terminal which basically was one of the first low chip count video terminals based around shift registers, my own design.

Filler: Why did you decide against an 8080 the second time around when you designed the Apple II?

Wozniak: The second time around we didn't have any successful products, we had no money. I owned the 6502. That was it. I could not afford an 8080, period.

Filler: What are your feelings looking back at the extended cycle design. Is that something that's no big deal or is it something that came in a flash of brilliance, or something that you don't feel happy about?

Wozniak: No, no flash of brilliance. It's just there's a problem with vertical strong color lines in NTSC video and it was just the obvious way to get around it. It was pretty easy to think of, and just about anybody trying to solve the same problem with the same circuit would have thought of it.

Filler: I heard you once talking about how you laid out the chips and traces for the Disk II controller board with an aesthetic level of design in mind. Did you get involved in it at that level on the Apple II motherboard itself?

Wozniak: No, it's too complicated to deal with on that level. We had a separate person being paid to lay out a PC board who knew how to do it. And, of course, it took him about two weeks full time. I pretty much helped by giving good ideas for which blocks of chips had the most interconnections to each other and where they should go on the board. I was involved to that extent as to what should be nearest to the bus and next over, and the next thing over. That was it.

Filler: There are a few real interesting circuit lines coming out on the //e auxiliary slot, like the one that looks like a clock enable line. Can you say anything about future plans for the auxiliary slot from Apple?

Wozniak: Believe it or not, I'm not that familiar with the auxiliary slot. I would say that beyond the Apple //e the auxiliary slot will never be used again, even in successors to the //e. I wish it hadn't have been used in the //e.

Filler: What do you think are the most interesting new chips coming out in terms of implications for microcomputers in the future?

Wozniak: Some of the 32 bit processors I think will have a future importance. I would say that the PAL design chips are extremely important because it's a flexible way to implement, very inexpensively, very high speed logic.

Filler: Do you think that 68000 boards for the II will have anything like the impact that Z-80 boards did, down the line once we have 68000 software out there?

Wozniak: Not really. I don't predict it.

Filler: So when people buy their second computer, are they going to buy another Apple II to stuff more cards into or are they going to go for a 68000 machine?

Wozniak: It depends on what's available. They will probably compare them and which one gives the most happiness in the end, which usually means which one solves the most problems for them and might be based around software they're already used to. They might buy a future Apple II, or, if they want some very high performance or whatnot that they can't get out of an Apple II, they might buy a 68000 machine. Or if there is some better, certain software they want on a different machine they may go for the other machine. It's an individual decision.

Filler: How about portability and CMOS chips and things like that for the II. Any thoughts that you can talk about?

Wozniak: I think we're always working on major directions today that are pretty obvious. And portable computers is one of them. In general, almost everybody these days is working on portable computers and integrated hard disk drives and built-in drives and portable displays, as well as lowering costs of products and building more enhanced products.

Filler: At the time when the Lisa was being designed, I guess people decided it wasn't really the right time yet to go for voice in a big way. Is that going to change soon?

Wozniak: I think pretty much in the business world the things that need solutions are so intricate with so many little specific details that need to be used that voice doesn't quite apply. You pretty much have to do a lot from the keyboard. For most home and mid-range products, voice output is very cheap. It costs almost nothing. That's generally worth something everywhere. Voice input is still . . . it's just going to be a long time before there's a lot of really good usable packages . . . some software package where it really works well and cheap.

Filler: What do you think the impact of ProDOS will be on Apple IIs?

Wozniak: I think it's very much worth watching. It's a very fine operating system. I wish we had it three years ago. I think a lot of more professional software will start to be written, and, of course, that puts a demand on us for other features, like more memory.

Chapter 43

John Sculley: President of Apple Computer

San Jose Municipal Airport is the world's gateway to California's "Silicon Valley." From there, you can take Stevens Creek Blvd. west into Cupertino, where Apple Computer's corporate headquarters is spread out amongst a number of buildings. The Triangle building on Stevens Creek Blvd. houses some of the engineering labs, the company's principal business address is at 20525 Mariani Ave., and the executive headquarters are in a second building on Mariani Ave.

Apple II engineers and system programmers in the Triangle building were very enthusiastic about Apple's new president, John Sculley. Six months after his arrival in May of 1983, they felt that much of the bureaucratic clutter of preceeding years was melting away. Apple II projects which had been postponed for years were now roaring along at full speed. Engineers felt their ideas were getting across to the top level management and marketing staff.

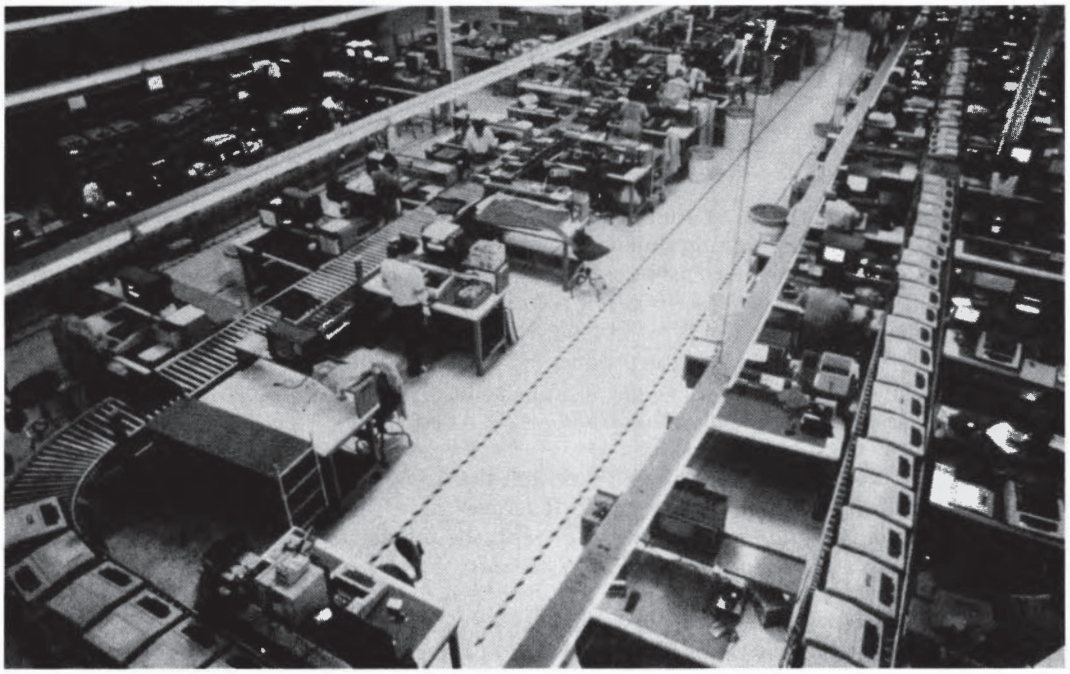


Fig. 43.1 Apple II assembly line.

Steven Jobs had wooed John Sculley away from PepsiCo. Jobs was interested in Sculley both because of his reputation as a marketing and organizational wizard and because of his long time interest in personal computers.

The following interview was conducted by Aaron Filler and Nick Anis at Mr. Sculley's office in late November of 1983.

Interview

Aaron Filler: First off, what do you see as some of the major challenges in marketing, particularly of the II, within your whole company's line of products?

Mr. Sculley: Well, the big challenge with the II is to keep the momentum going at a time when the market is being bombarded with so many products from so many different vendors. On the one hand, some people are not sure what the technology measurement is that they should judge the II by and, on the other hand, the success that IBM has had in the business market with the IBM PC and XT. So it raises two issues with the II. One is the price value of the product relative to this bombardment of hundreds of new products being offered by the Japanese and start-up companies and other established companies.

The value of the IBM PC and XT is not a price issue, it's more that MS-DOS is becoming one of the major standards in the industry. So that means for us from a strategy standpoint we have to figure out how do we build more performance into the Apple II and how do we determine what the right price is that we can charge for that performance to establish a value for the product so that we can continue to maintain the Apple II as a strong mainstream product going forward. And we intend to do that by doing three things:

One is to add peripherals and accessory products to it, which we are in the process of doing right now—things like the mouse, which we brought down from the LISA technology area, which Woz worked on; ProDOS, which gives us access to a hard disk; some integrated software—a new one called AppleWorks, which has a data base word processing and spreadsheet in a single application, and we are working on new products which will have more performance in terms of speed, in terms of memory, and things like that.

Secondly, we want to recognize that the Apple II has a user base that is probably different than any other personal computer, including the IBM PC. That user is really loyal, incredibly loyal to Apple, and expects loyalty in return. The loyalty in return is that we've got to give people who already own Apples, even if we don't make any money on it, we've got to give people who already own Apples some gateway that they can benefit from the newer technologies that we are able to bring to the product. So as we develop new products that are higher performance for the Apple II, we've got to have some way that there can be board swaps or something that can allow people to upgrade.

The third thing is that we want to make sure that the Apple II is not looked at as the industry standardizing to MS-DOS and then Apple II is over here with its own unique proprietary operating system in some niche. We want to be in the mainstream and the way to do that is we've got to have a family of products.

We're looking at other products that we can target against different markets. For instance, the Apple II is an all-purpose machine. It works in the education market, serious home use, and business, and what we want to do is make sure that we have a family of products we can

target specifically towards what does the education market need, what does the home market need, what does business need. We'll probably have more than a single Apple II in the future, but we're not walking away from it. And that's the whole story.

Filler: In that answer you anticipated most of my other questions, so I can move over to a different area. What are your challenges in terms of encouraging good engineers and programmers to come to Apple?

Sculley: I think the key challenge is the environment, the culture that we have at the company, which is reflected by what we call "Apple values." As Apple gets larger and as Apple changes as the industry changes, what we want to be very careful of is that we don't just look like every other vanilla corporation. People came to Apple in the first place because they didn't want to be in one of these other vanilla corporations. They came here, we believe, because they felt that there wasn't a lot of hierarchy and bureaucracy and that engineers could go and build products that they would want to own themselves and that these products would really get out in the market in a reasonable period of time.

Many people came here because they had dreams of products that they couldn't get other companies to build. What we can't do is just go off and arbitrarily build products, we'll just end up with products which have different operating systems and no compatibility and we won't be a successful company. So within the parameters of some strategy, we want to have the environment of small teams without a lot of bureaucracy, where engineers can actually take a product or a project, as the case may be, and move it all the way through and bring it to the market in a pretty short period of time. And instead of taking three and a half years, which is about what it's taken us to develop products, we're trying to do it in about a year's time, and that's real hard.

Filler: Are there any major new directions in research and development that you can talk about that we haven't seen in the current product line.

Sculley: Well, there are a lot of things you haven't seen but we're not prepared to talk about them.

Nick Anis: There are many companies that allow and encourage purchase of computers through the mail. In the near future will Apple be coming out with a mass merchandising product and a lower priced Apple II.

Sculley: We've never said we're coming out with a mass merchandise product.

Anis: Does Apple ever see itself having products that can be purchased through the mail for people who live in more isolated areas.

Sculley: We have no plans at this time to go through mail order. Never is a long time. But our main interest is not to make the mistakes that say, TI, did by driving the dealer margins out not only for themselves but for the dealer on the CPUs, for accessories, and for peripherals and for software so that nobody made any money. We don't want to make the mistake that Sony did, which was to take a really high quality company with great quality products and then to spread themselves so broadly that people had used it as a traffic builder and drove the margins out and nobody could make any money. Then Sony got in trouble. So we want to be really careful how we expand distribution and we want to be really careful that we don't try to turn Apple into a low end game company, because that's not where we want to be.

Anis: Market penetration is reported at 12 percent. How do you feel about market penetration?

Sculley: I've never seen the 12 percent figure; I've seen a seven percent figure.

Anis: Do you feel that's accurate?

Sculley: I don't know and nor do I care. I think it's a totally irrelevant way to look at the industry. I think most of the projections that say it's going to be a $90 billion industry or a $150 billion industry are all based on extending out trends and looking at penetration figures. I think far more significant in this industry than penetration are really milestones.

The Apple II was a milestone when it took high cost technology and translated it into a low-cost, easy to use, reliable product. VisiCalc was a milestone—you could do something useful with an Apple II. The IBM PC was a milestone. The credibility of IBM. Lotus 1-2-3 was a milestone—you could do something real useful with the IBM PC. I think that it will be a series of milestones, primarily within the parameters of making the products easier and easier to use and making them do more and more useful things that's going to determine how large the industry is going to be.

Texas Instruments, I think, made a classic error by assuming that going out and buying market share by dropping price is going to increase penetration and therefore when you had market share you could eventually, as technology cost came down, hold your price and make money. And it doesn't work in this industry because products that don't do much aren't a value at any price. Obviously penetration will go up, but rather than try to guess what the penetration figures are, we're much more interested in trying to peg what the milestones are going to be.

V

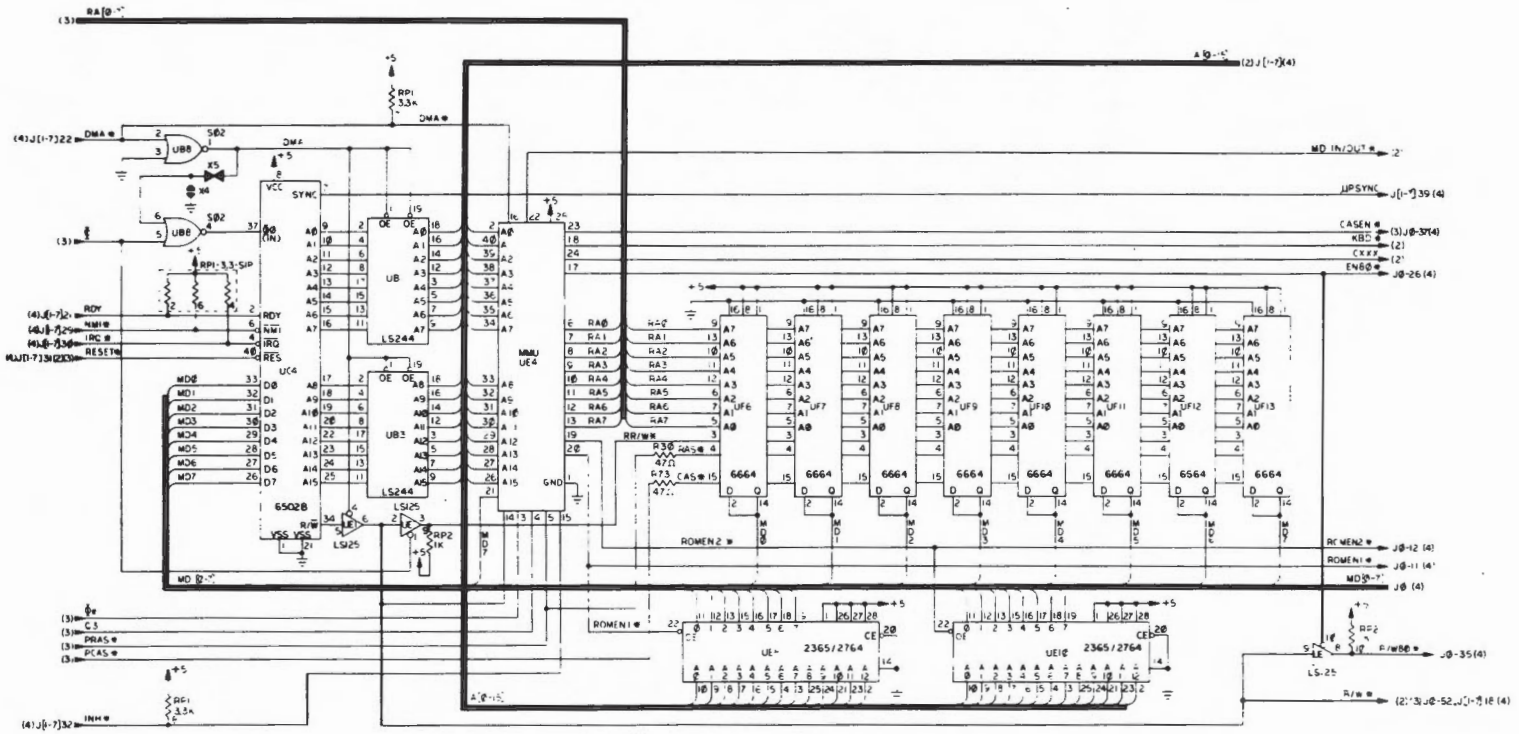
APPENDICES

Appendix A

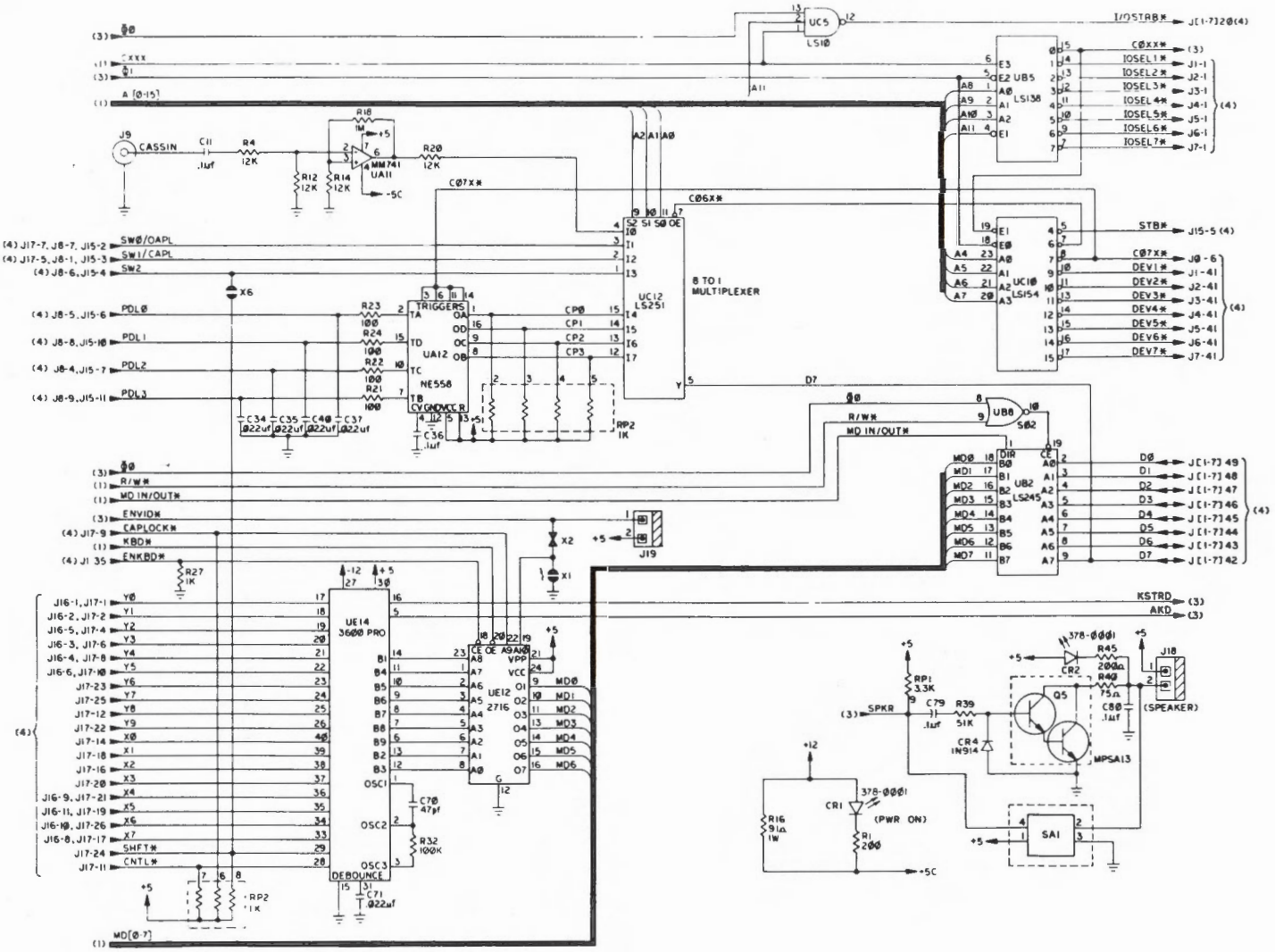
Apple //e and Apple //c Schematics

Apple //e

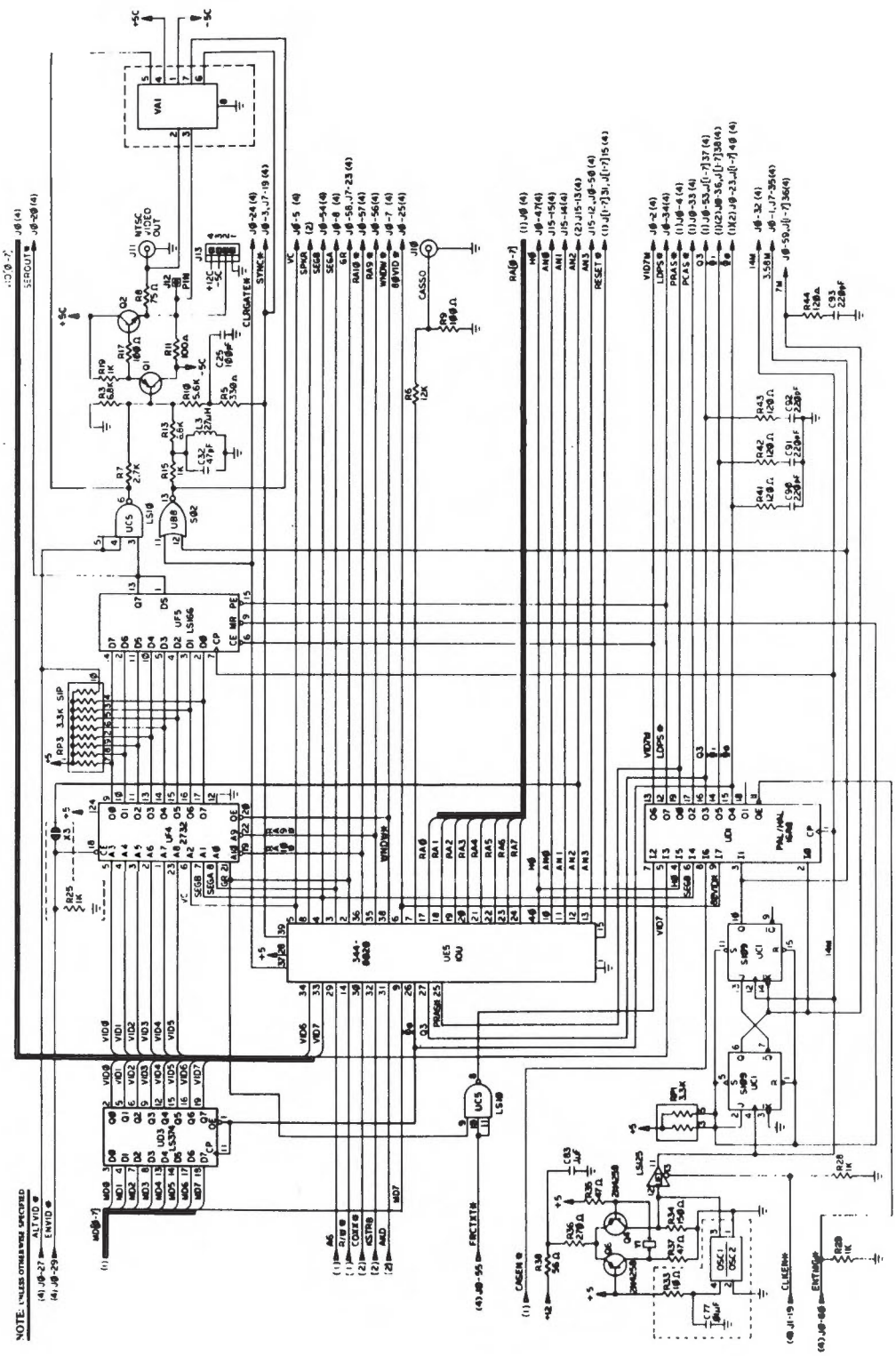
Schematic Diagram,
part 1



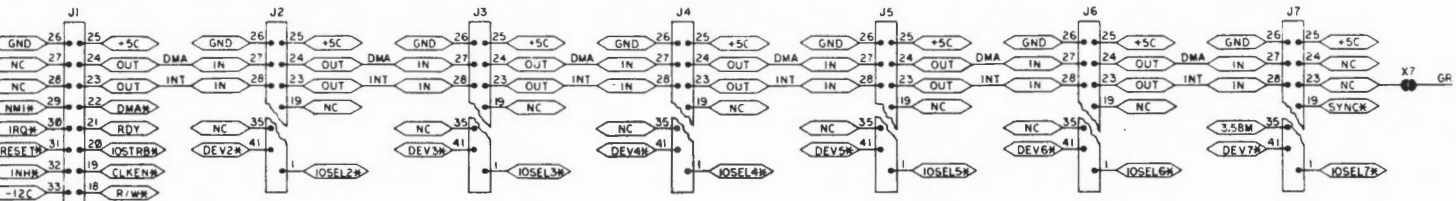
ematic Diagram,
t 2

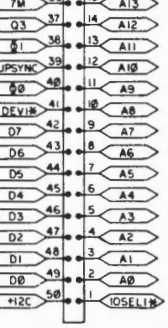


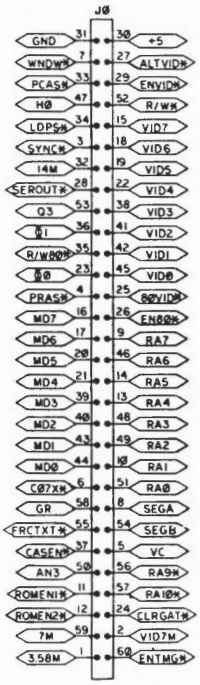
Schematic Diagram,
part 3

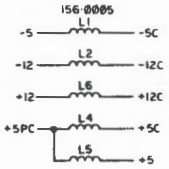


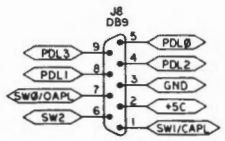
chematic Diagram,
art 4

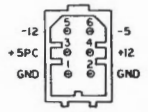


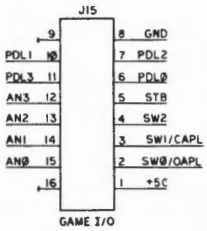


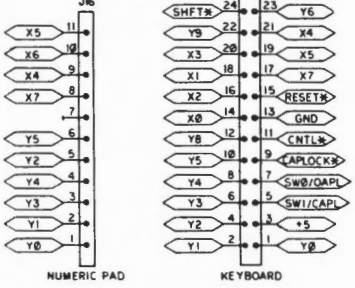






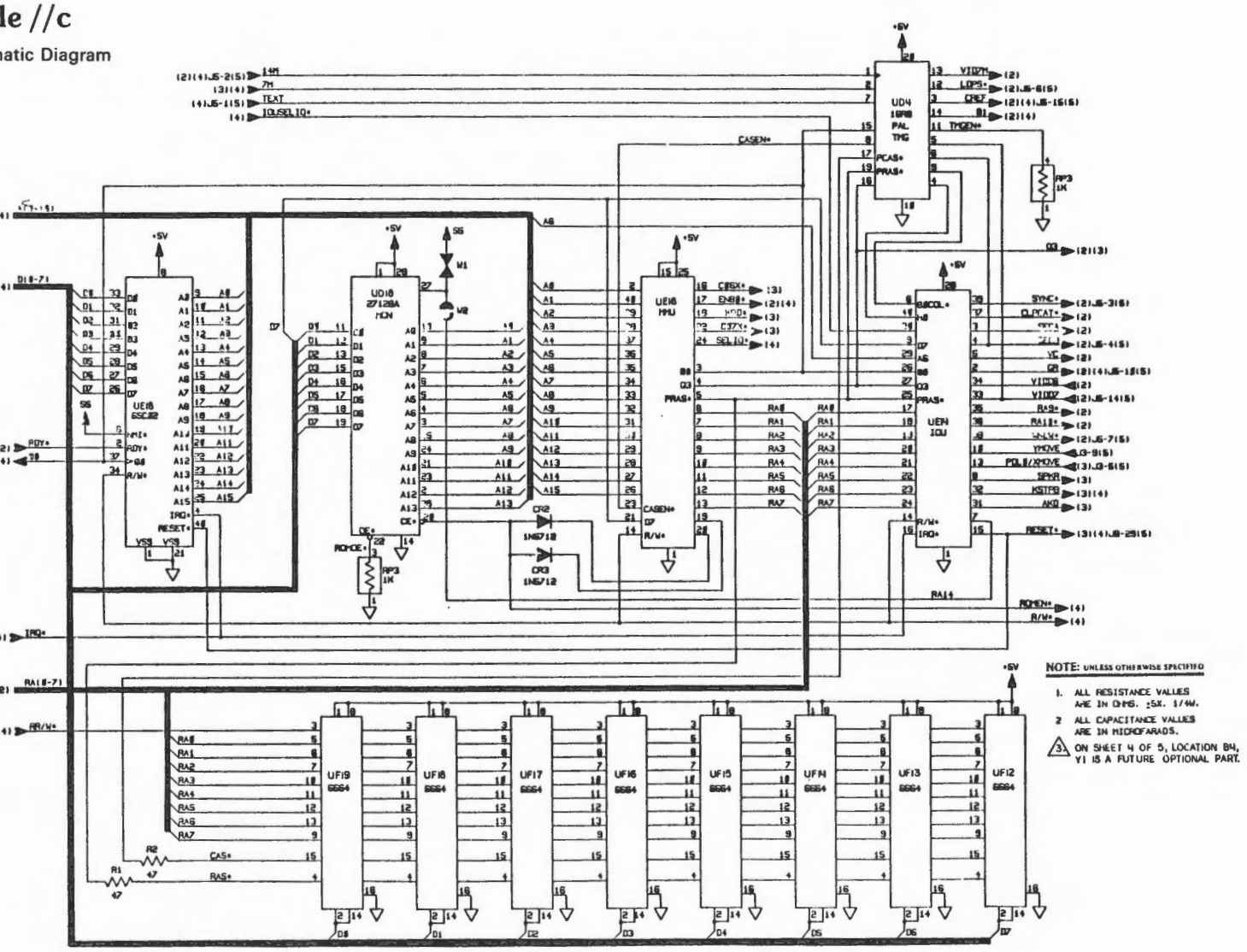






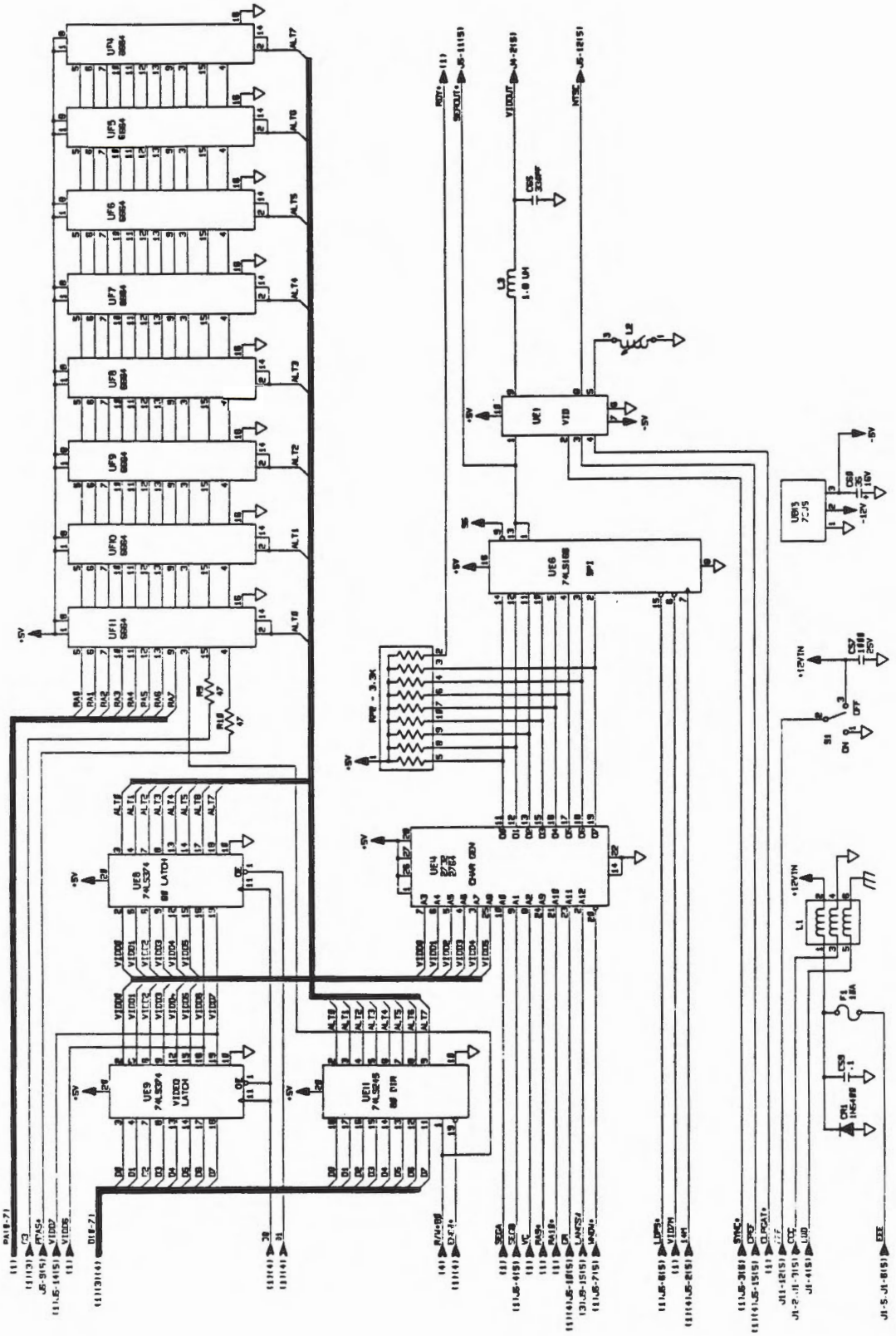
le //c

natic Diagram

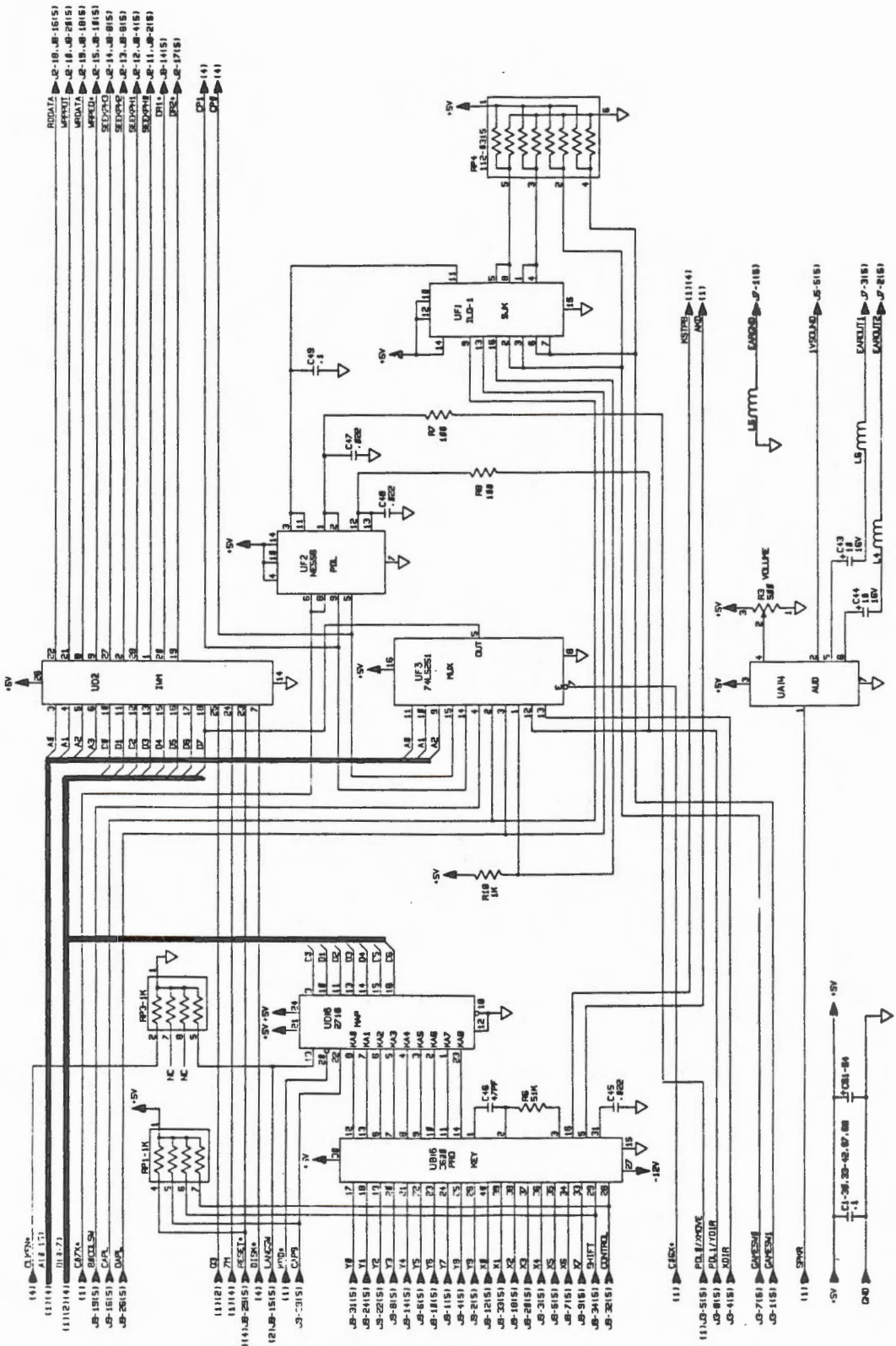


Schematic Diagram

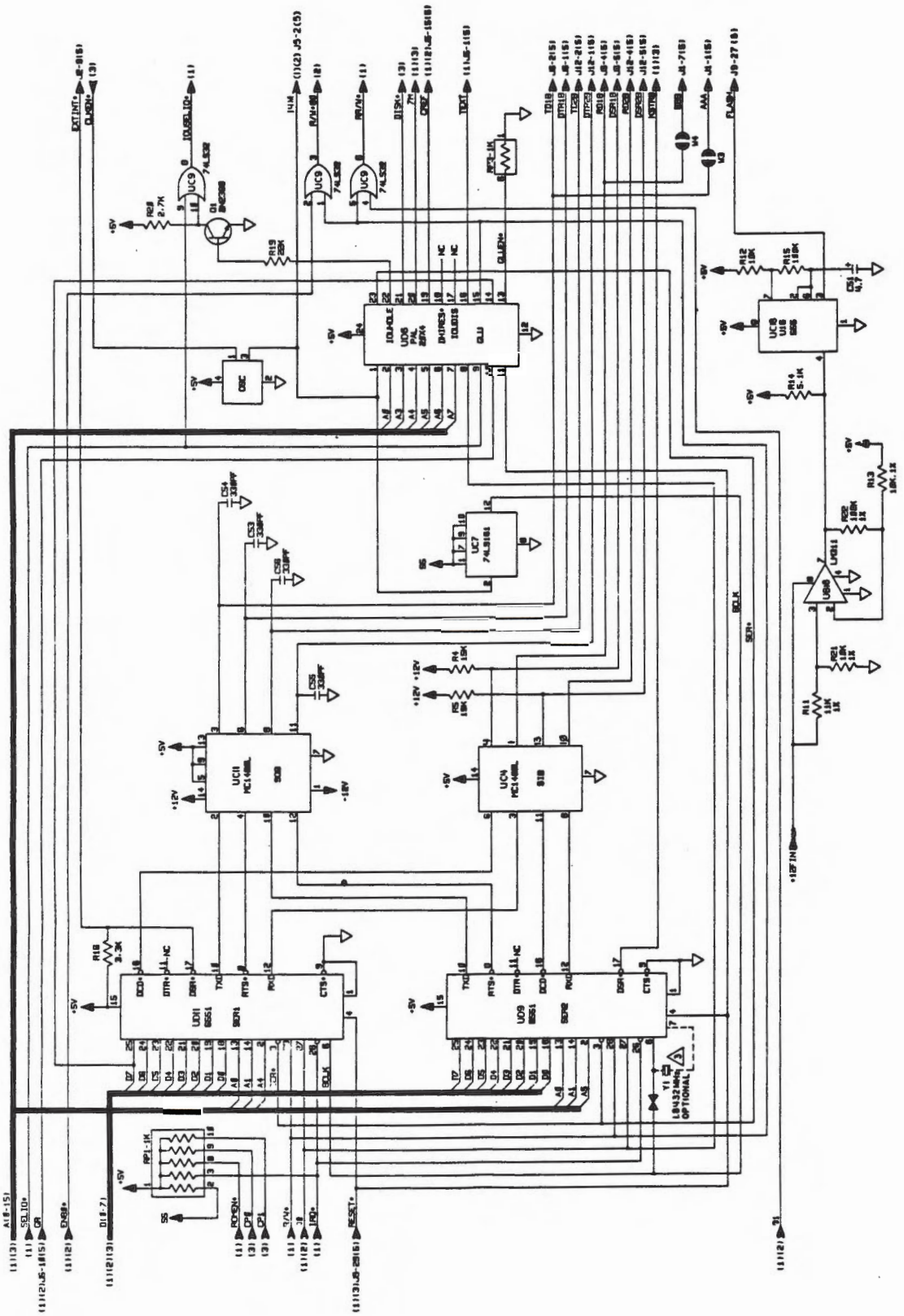
part 2



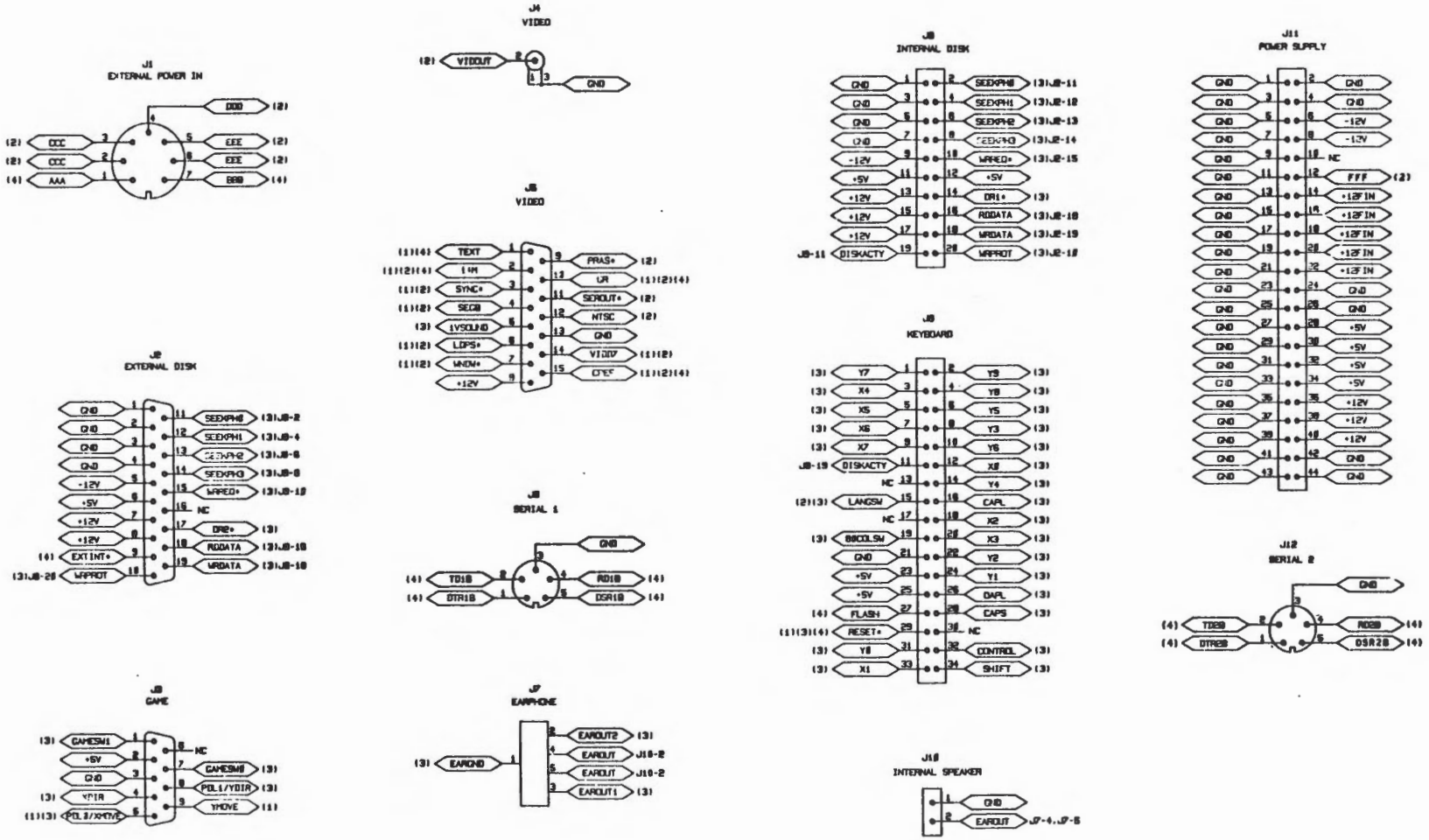
Schematic Diagram
part 3



Schematic Diagram
part 4



Schematic Diagram
part 5



Appendix B

Apple II Product Specifications

Hobby Prototyping Board

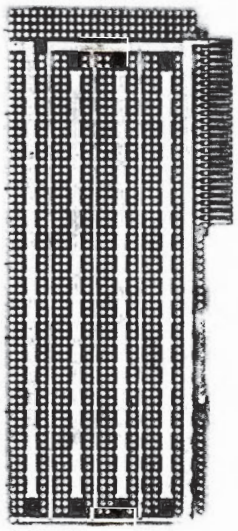
GENERAL DESCRIPTION

The APPLE II Hobby/Peripheral Board is a 2-3/4" x 7" two sided printed circuit board with a hole pattern on 100 mil centers. The board will accept all conventional integrated circuit packages as well as transistors, resistors, capacitors and other passive components. It is intended for use as a means of breadboarding experimental circuitry or a unique interface for a computer peripheral or hobby accessory.

FEATURES

- Row and Column Alpha Numeric identification for easy layout reference
- Heavy duty power and ground bus
- Hole pattern on 100 mil grid for any size IC
- Gold Plated contacts
- Computer grade board
- Wire wrap or point to point interconnect

ORDERING INFORMATION Order Product Code A2B0001X



Board Photograph

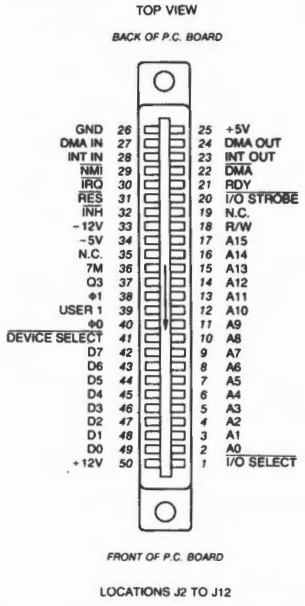


Figure 1. Peripheral Connector Pinout

TABLE I

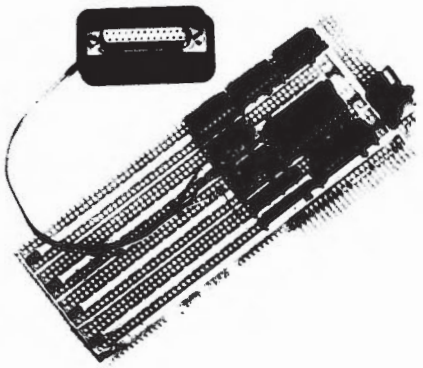
SIGNAL DESCRIPTION FOR PERIPHERAL I/O CONNECTORS

Pin #	Pin Name	Description
1	I/O SELECT	256 addresses are set aside for each peripheral connector (see address map). A read or write at such an address will send Pin 1 on the selected connector low during ϕ_0 (500ns). This line is not available on peripheral connector 0. Refer to Table II for address allocation.
2-17	A_0-A_{15}	16-bit system address bus. Addresses are set up by the 6502 within 300ns after the beginning of ϕ_1.
18	R/$\overline{W}$	READ/$\overline{WRITE}$ line from 6502. When high indicates that a read cycle is in progress, and when low that a write cycle is in progress.
19	NC	No connection.
20	$\overline{I/O\ STROBE}$	Pin 20 on all peripheral connectors will go low during ϕ_0 of a read or write to any address $C800-$CFFF. Refer to Table II for address allocation.
21	RDY	"Ready" line to the 6502. This line should change only during ϕ_1, and when low will halt the microprocessor at the next READ cycle. This line has a 3KΩ pullup to +5V.
22	$\overline{DMA}$	Direct Memory Access control output. This line has a 3KΩ pullup to +5V.
23	INT OUT	Interrupt daisy chain output to lower priority peripheral devices. Note: Pins 28 and 23 must be connected together if not used on the card.
24	DMA OUT	Direct Memory Access daisy chain output to lower priority peripheral devices. Note: Pins 27 and 24 must be connected together if not used on the card.
25	+5V	Positive 5-volt supply*, 500 mA total for all peripheral boards together.
26	GND	System circuit ground. 0 volt line from power supply.
27	DMA IN	Direct Memory Access daisy chain input from higher priority peripheral devices. Usually connected to Pin 22.
28	INT IN	Interrupt daisy chain input from higher priority peripheral devices. Usually connected to Pin 23.
29	NMI	Non Maskable Interrupt request line to the 6502. This line has a 3KΩ pullup to +5V. It is active low.
30	$\overline{IRQ}$	Interrupt request line to the 6502. This line has a 3KΩ pullup to +5V. It is active low.
31	$\overline{RES}$	Reset line from "RESET" key on keyboard. Active low.
32	$\overline{INH}$	Inhibit Line. When a device pulls this line low, all ROM's on board are disabled (Hex addressed $D000 through $FFFF). This line has a 3KΩ pullup to +5V.
33	-12V	Negative 12 volt supply*, 200mA total for all peripheral boards together.
34	-5V	Negative 5 volt supply*, 200 mA total for all peripheral boards together.
35	NC	No connection.
36	7M	Seven MHz high frequency clock.
37	Q_3	A 2MHz (nonsymmetrical) general purpose timing signal.
38	ϕ_1	Phase 1 clock, complement of ϕ_0 clock.
39	USER 1	The function of this line will be described in a later document.
40	ϕ_0	Microprocessor phase ϕ_0 clock.

Table I (Continued)

Pin #	Pin Name	Description
41	$\overline{\text{DEVICE SELECT}}$	Sixteen addresses are set aside for each peripheral connector. A read or write to such an address will cause Pin 41 on the selected connector to go low during ϕ_0 (500ns). Refer to Table II for address allocation.
42-49	D_0-D_7	8-bit system data bus. During a write cycle, data is set up by the 6502 less than 300ns after the beginning of ϕ_2. During a read cycle the 6502 expects data to be ready no less than 100ns before the end of ϕ_0.
50	+12V	Positive 12 volt supply*, 250mA total for all peripheral boards together.

*Note: Total power drawn by any one peripheral board is not to exceed 1.5 watts.



Typical Hobby Board Configuration

DESIGN TECHNIQUES FOR APPLE II HOBBY BOARDS

GENERAL LAYOUT

Figure 1 is an illustration of an APPLE II peripheral board in an I/O connector. Note that the short end of the board is to the back edge of the system and components are mounted on the side containing the Apple Computer logo. The APPLE II hobby boards are designed to have both +5V and GND available on both sides of the board. If other voltages are used they must be individually wired. The boards are designed for use with either wire-wrap or conventional wiring techniques. IC sockets are generally recommended in hobby applications. TTL should be low power Schottky where possible.

DECOUPLING

All voltages used on the hobby board should be decoupled with a 0.1μf capacitor to ground near the I/O connector board power pin. Additional 0.1μf capacitors should be used for approximately every four low power Schottky, CMOS, or MOS devices. DO NOT USE HIGH VALUE ELECTROLYTIC DECOUPLING CAPACITORS. They can cause improper operation of the APPLE II power supply.

If PROM or buffer power down is used, the power down circuit should be individually decoupled. Do not decouple the switched power pin.

I/O LOADING AND DRIVE RULES

Table II gives the drive and loading requirements for the peripheral I/O connector. The address bus, the data bus, and the R/W lines should be driven by tri-state buffers. The 74LS365 is typically a good choice. Remember that there is considerable capacitance distributed over the board, and one should look forward to tolerating the extra load of seven other peripheral cards. Attempting to use PIA's and ACIA's directly on the bus will generally lead to timing and level errors. Type 2316 ROM's are an exception because the timing allows a very large margin. The drive required and the maximum loading allowed are stated in terms of low power Schottky logic (LSTTL).

TABLE II
LOADING AND DRIVING RULES

Pin Number	Name	Required Drive	Maximum LSTTL Load
1	$\overline{\text{I/O SELECT}}$	N/A	10
2-17	A_0-A_{15}	Tri-State Buffer	5
18	R/$\overline{W}$	Tri-State Buffer	10
19	N/C	N/A	N/A
20	$\overline{\text{I/O STROBE}}$	N/A	2
21	RDY	Open Collector	N/A
22	$\overline{\text{DMA}}$	Open Collector	N/A
23	INT OUT	4 LSTTL	N/A
24	DMA OUT	4 LSTTL	N/A
25	+5V	N/A	N/A
26	GND	N/A	N/A
27	DMA IN	N/A	4
28	INT IN	N/A	4
29	NMI	Open Collector	N/A
30	IRQ	Open Collector	N/A
31	$\overline{\text{RES}}$	N/A	2
32	$\overline{\text{INH}}$	Open Collector	N/A
33	-12V	N/A	N/A
34	-5V	N/A	N/A
35	N/C	N/A	N/A
36	7M	N/A	2
37	Q3	N/A	2
38	ϕ_1	N/A	2
39	USER 1	N/A	N/A
40	ϕ_0	N/A	2
41	$\overline{\text{DEVICE SELECT}}$	N/A	10
42-49	D_0-D_7	Tri-State Buffer	1
50	+12V	N/A	N/A

TIMING SIGNALS

A number of system timing signals are available on the APPLE II Bus. Figure 2 shows the details of the relative timing of the available signals. ϕ_0 on the APPLE II Bus has significant on-board fan-out and should not be used on APPLE II peripherals. Inverting ϕ_1 will provide a safe ϕ_0 signal.

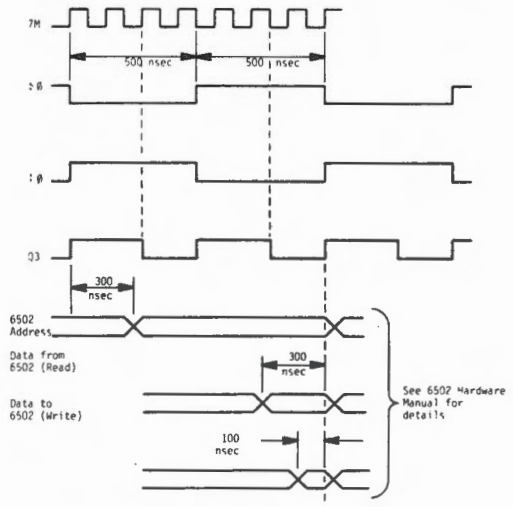


Figure 2. 6502 Timing Diagram

TABLE III
PERIPHERAL ADDRESS ALLOCATION

Hex Address	Assigned Function	Slot Number	Comments
C080-C08F	$\overline{\text{DEVICE SELECT}}$	0	Pin 41 on the selected Peripheral Connector goes low during ϕ_0.
C090-C09F	$\overline{\text{DEVICE SELECT}}$	1	
C0A0-C0AF	$\overline{\text{DEVICE SELECT}}$	2	
C0B0-C0BF	$\overline{\text{DEVICE SELECT}}$	3	
C0C0-C0CF	$\overline{\text{DEVICE SELECT}}$	4	
C0D0-C0DF	$\overline{\text{DEVICE SELECT}}$	5	
C0E0-C0EF	$\overline{\text{DEVICE SELECT}}$	6	
C0F0-C0FF	$\overline{\text{DEVICE SELECT}}$	7	
C100-C1FF	$\overline{\text{I/O SELECT}}$	1	Pin 1 on the selected Peripheral Connector goes low during ϕ_0.
C200-C2FF	$\overline{\text{I/O SELECT}}$	2	NOTE: Peripheral Connector 0 does not get this signal.
C300-C3FF	$\overline{\text{I/O SELECT}}$	3	
C400-C4FF	$\overline{\text{I/O SELECT}}$	4	
C500-C5FF	$\overline{\text{I/O SELECT}}$	5	
C600-C6FF	$\overline{\text{I/O SELECT}}$	6	
C700-C7FF	$\overline{\text{I/O SELECT}}$	7	
C800-CFFF	$\overline{\text{I/O STROBE}}$	ALL	I/O Common Address Space. Pin 20 or all peripheral connectors go low during ϕ_0.

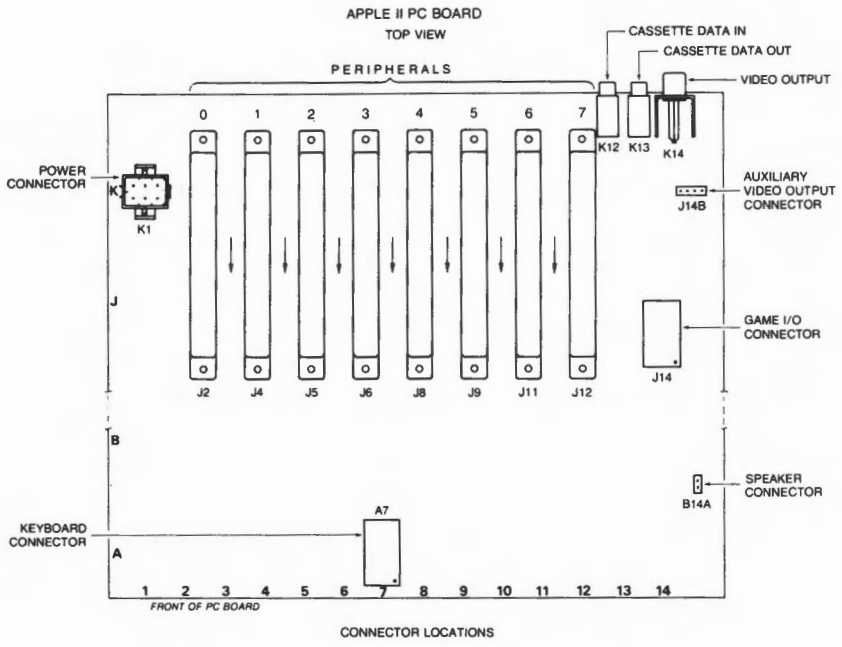


Figure 3. Connector Location Detail

I/O PROGRAMMING SUGGESTIONS

It is usually necessary for the program on an I/O board in $CNXX space use its I/O slot number for indexing into the $C080 + $N0 DEVICE SELECT space and to use the slot dependent RAM locations. The program can determine its own location by using a dummy RTS and getting the return location from the stack. The following is an example of the code:

```
CN08 08          PHP          Save Status
CN09 78          SEI          Disable Interrupt to
                                preserve stack
CN0A 20 58 FF    JSR IORTS    Dummy RTS at FF58
CN0E BD 00 01    TSX          Load stack pointer
CN11 28          LDA PAGE1,X  Load slot# in accumulator
                 PLP          Restore Status
```

The slot number $CN will now be in the accumulator and may be shifted to form $N0 or masked to form $0N. With $N0 in the X register, the DEVICE SELECT addresses may be referenced by:

LDA $C080,X

NOTE: The 6502 will cause a read cycle twice at location $C080 + $N0. The first of these is a false read. Simiarly a store cycle at location $X080 + $N0 will cause a false read cycle followed by the write cycle. These false read cycles can disturb the status register of peripheral devices such as PIA's or ACIA's. Study the 6502 Programming Manual for details on indexed memory operations.

Similarly, with $0N in the Y register, a slot dependent RAM address may be referenced by:

LDA $0478,Y

For this example, if we're dealing with a peripheral in slot 2, the address referenced is $47A. In the same manner, we may access $4FA, $57A, etc. The standard system character input and output subroutines are vectored through switches in RAM. CSW (locations $36 low and $37 high) contains the two-byte address of the current character output subroutine, normally $FDF0 for the APPLE II video display. KSW (locations $38 low and $39 high) contains the two-byte address of the

current character input subroutine, normally $FD1B. Either of these may be changed by the user for dual I/O applications. For example, setting location $36 to $80 and location $37 to $03 will set the current character output entry point as $0380.

Standard character input and output subroutine entry points have been assigned to peripheral slots #1-7:

Slot	Entry
1	$C100
2	$C200
3	$C300
4	$C400
5	$C500
6	$C600
7	$C700

The standard character I/O subroutine entry points may be conveniently placed in CSW (output) and KSW (input) with the BASIC commands PR#N and IN#N respectively, where N is the appropriate slot number. The commands are NP^C (P^C is "control P") and NK^C from monitor. For example, the BASIC command PR #3 sets CSWL (locations $36 and $37) to $C300.

The character output subroutine is passed one character at a time in the A register. On the APPLE II, the parity bit (b7) shall normally be set. This subroutine must return with the A-, X-, and Y- registers, undisturbed, and the decimal mode flag clear for existing software to function.

The character input subroutine expects one character to be passed in the A register. The parity bit should be set to work properly with existing hardware.

Apple Computer has adopted certain conventions pertaining to the use of peripheral select and strobe lines and memory addressing schemes. It is advised that these conventions be reviewed particularly when it is expected that more than one peripheral card will be installed.

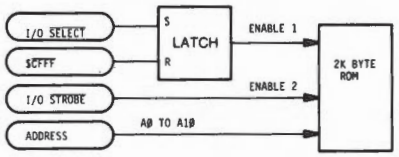
The following discussion applies to peripheral slots number 1 through 7. Slot number 0 is excepted. (See Figure 3).

Pin 1 of the peripheral connector is the I/O SELECT line. This line is active low. It is normally high, and goes low whenever the address bus carries $CNXX, when N equals the peripheral slot number. For example: looking at the peripheral connector number 4, Pin 1 goes low when the address bus reads $C400, or $C401, or $C402 or any hex number from $C400 up through $C4FF. The transition from high to low takes place as ϕ_1 clock line goes low (plus the prop delays through the LS138 decoding logic at F-12 and H-12). One intended use for this line is the chip select for the peripheral card's PROM.

Pin 41 of the peripheral connector is the $\overline{\text{DEVICE SELECT}}$ line. This line is active low, and is derived from the same decoding chain as the I/O SELECT line in an additional LS138 (located at H-2). $\overline{\text{DEVICE SELECT}}$ is normally high, and goes low whenever the address bus carries $C0MX, where $M is 8 plus the peripheral slot number (i.e.: $M = $8 + $N). For example, Pin 41 goes low at peripheral connector number 4 for addresses $C0C0 through $C0CF and for connector number 5, Pin 41 goes low for addresses $C0D0 through $C0DF, and so on. The intended use for this line is the chip select for the peripheral card's ACIA, PIA and other "controller" IC's.

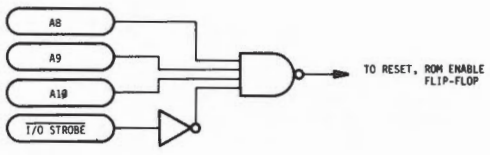
Pin 20 of the peripheral connector is the $\overline{\text{I/O STROBE}}$ line, and is common to all the peripheral connectors. This line is active low and is derived from the LS138 (located at F-12). All peripheral Pin 20's go low when the address bus carries any address within the range $C800 to $CFFF, inclusive. The intended use of this line is for ROM and PROM expansion, and the address $CFFF is allocated to disable the expansion scheme. The following diagram represents such a scheme.

Figure 4.



To simplify decoding the ROM enable RESET, the following scheme can be implemented. By giving up access to 256 bytes out of the 2K bytes of the ROM and using the $\overline{\text{I/O STROBE}}$ line, the hardware requirement is reduced to the following:

Figure 5.



PERIPHERAL SCRATCHPAD MEMORY

Certain areas of RAM are set aside for peripheral RAM storage. The specific addresses are shown in the table below (all are hex).

Location $7F8 is a special one. This location should be loaded with $CN, where N is the slot number of the active peripheral, whenever an interrupt may occur and the ROM/PROM expansion scheme is in use. This is necessary so that the return from interrupt software used allows the proper peripheral card to resume operation.

Common Scratchpad Address	Peripheral Slot Number						
	1	2	3	4	5	6	7
478	479	47A	47B	47C	47D	47E	47F
4F8	4F9	4FA	4FB	4FC	4FD	4FE	4FF
578	579	57A	57B	57C	57D	57E	57F
5F8	5F9	5FA	5FB	5FC	5FD	5FE	5FF
678	679	67A	67B	67C	67D	67E	67F
6F8	6F9	6FA	6FB	6FC	6FD	6FE	6FF
788	779	77A	77B	77C	77D	77E	77F
7F8*	7F9	7FA	7FB	7FC	7FD	7FE	7FF

Appendix C

Auxiliary Slot Signals

Pin Number	Name	Description
1	3.58M	3.58 MHz video color reference signal. This line can drive two LS TTL loads.
2	VID7M	Clocks the video dots out of the 74166 parallel-to-serial shift register. This line can drive two LS TTL loads.
3	SYNC'	Video horizontal and vertical sync signal. This line can drive two LS TTL loads.
4	PRAS'	Multiplexed RAM row-address strobe. This line can drive two LS TTL loads.
5	VC	Third low-order vertical-counter bit. This line can drive two LS TTL loads.
6	C07X'	Hand-control reset signal. This line can drive two LS TTL loads.
7	WNDW'	Video non-blank window. This line can drive two LS TTL loads.
8	SEGA	First low-order vertical counter bit. This line can drive two LS TTL loads.
51, 10, 49, 48, 13, 14, 46, 9	RA0-RA7	Multiplexed RAM-address bus. This line can drive two LS TTL loads.

Pin Number	Name	Description
11, 12	ROMEN1, ROMEN2	Enable signals for the ROMs on the main circuit board.
15	R/W'	Read/write signal from 6502. This line can drive two LS TTL loads.
44, 43, 40, 39, 21, 20, 17, 16	MD0-MD7	Internal (unbuffered) data bus. This line can drive two LS TTL loads.
35	R/W80	Read/write signal for RAM on the 80-column text card. This line can drive two LS TTL loads.
36	ϕ1	6502 clock phase 1. This line can drive two LS TTL loads.
37	CASEN'	Column-address enable. This signal is disabled (held high) during accesses to memory on the auxiliary card. This line can drive two LS TTL loads.
47	H0	Low-order horizontal byte counter. This line can drive two LS TTL loads.
50	AN3	Output of annunciator number 3. This line can drive two LS TTL loads.
52	R/W'	6502 read/write signal. This line can drive two LS TTL loads.
53	Q3	2 MHz asymmetrical clock. This line can drive two LS TTL loads.
54	SEGB	Second low-order vertical-counter bit. This line can drive two LS TTL loads.
55	ENFIRM	Normally high; pulling this line low disables ROM1 and ROM2 on the main circuit board. This line has a 3300-ohm pullup resistor to +5V.

Pin Number	Name	Description
56, 57	RA9', RA10'	Character-generator control signals from the IOU. This line can drive two LS TTL loads.
58	GR	Graphics-mode enable signal. This line can drive two LS TTL loads.
59	7M	7 MHz timing signal. This line can drive two LS TTL loads.
60	ENTMG'	Normally low; pulling this line high disables the master timing from the PAL. This line has a 1000 ohm pulldown resistor to ground.
45, 42, 41, 38, 22, 19, 18, 15	VID0-VID7	Video data bus. This three-state bus carries video data to the character generator.
23	ϕ0	6502 clock phase 0. This line can drive two LS TTL loads.
24	CLRGAT'	Color-burst gating signal. This line can drive two LS TTL loads.
25	80VID'	Enables 80-column display timing. This line can drive two LS TTL loads.
26	EN80'	Enable for auxiliary RAM. This line can drive two LS TTL loads.
27	ALTVID'	Alternative video output to the video summing amplifier.
28	SEROUT'	Video serial output from 74166 parallel-to-serial shift register.
29	ENVID'	Normally low; driving this line high disables the character generator such that the video dots from the shift register are all high (white), and alternative video can be sent out via ALTVID'. This line has a 1000 ohm pulldown resistor to ground.
30	+5	+5 volt power supply.

Pin Number	Name	Description
31	GND	System common ground.
32	14M	14.3 MHz master clock signal. This line can drive two LS TTL loads.
33	PCAS'	Multiplexed column-address strobe. This line can drive two LS TTL loads.
34	LDPS'	Strobe to video parallel-to-serial shift register. This signal goes low to load the contents of the video data bus into the shift register. This line can drive two LS TTL loads.

Appendix D

Versions, Revisions, and Modifications

Apple I

This was designed by Wozniak as an independent project while he was an engineer at Hewlett Packard during 1975. There are a few Apple I's still in use and they are all most-certainly collectors items (see Chapter 42, Figure 42.1).

Apple II

Wozniak completed the Apple II design during 1976, along with Allen Baum who designed the slot system. (Allen's father Elmer was one of the original financial backers, and Allen's brother Peter handles a wide variety of technical support and design problems for Apple. Peter has been one of the advocates for removing the slots from the Apple //c and other later Apple II family products.)

The original motherboard has the part number 820-0001-XX, wherein the XX carries the revision number. The earliest models don't have a part number, the later models have the part number written under the 6502 on the board, and the latest models the part number ended up over on the left near the power supply.

Revision 00 is mostly to do with the time between the original design and the start of shipments in June of 1977. This early model can only get four colors in the hi-res mode because it can't use the high bit of the graphics data byte to get a time shift (see Chapter 7), and it just sits there being confused when you first turn it on because it doesn't have an automatic "power-on reset."

All told, there were about 40,000 Apple II's shipped, mostly with revision numbers 01, 02, and 03. These all include the full 6 true colors in hi-res graphics, automatic power-on reset, an optional 50 Hz video mode for European Apples, and "color killer" circuitry to help get clean white text characters on a color television or monitor.

Apple II+

The Apple II+ is really an upgrade of the system firmware stored in ROM. The Monitor has several minor changes noticeable only to true hackers, and one big change noticeable to every one. When you turn on an Apple II+, it automatically tries to turn on the disk drive and read something. This was a great leap in "user friendly" design. Some might argue that this doesn't compare favorably with later improvements such as the mouse or windows, but after all it was only 1979. This change gives the Monitor a new name, "The Autostart ROM," and it creates the Apple II+.

At the same time, Apple switched to "Applesoft BASIC." The old "Integer BASIC" did high speed math directly in the 6502, but it only worked in integers. Applesoft includes a large and high powered "floating point math package" (see Chapter 38). You can upgrade an Apple II to an Apple II+ by inserting the Autostart and Applesoft ROMs.

The first 100,000 or so Apple II+'s were shipped with the Revision 04 version of the original motherboard. It differs, mainly in that it is pretty much permanently configured to handle only 16K RAM chips, while the older versions could use 4K chips as well (see Chapter 21). However, the remaining half million Apple II+'s shipped between the Spring of 1981 and February 1983 all include a number of hardware changes in the motherboard and in the keyboard system as well as the firmware changes.

The Revision 7 and RFI Motherboards

None of the changes in the Revision 7 motherboard cause any noticeable effect for most users. However, the most important was a complete redesign to meet FCC regulations for reduced emission of Radio Frequency Interference (RFI). There are some Revision 7 motherboards without the RFI change (820-0001-07), but most have the RFI changes as well and have the new part number: 820-0044-XX wherein XX has the revision number 01, C, or D. The easiest change to spot in the disappearance of the three "RAM configuration blocks" (see Chapter 21, Figure 21.8); these were no longer needed since all Apple II+'s were shipped with three rows of 16K RAM chips.

A few chip changes of modest consequence include some improved "color killer" circuitry, and a switch from two "8T28 quad transceivers" to a single "8304 octal transceiver" to connect the 6502 to the system data bus. This or some related change greatly improves the reliability of the "DMA" system (see Chapter 28) so it's an important concern for users of the Vista A800 floppy disk controller.

Another fairly important circuit change makes it possible to use a "2316" ROM for the character generator instead of the old "2513." This is important because it's easy to program ROM's that are compatible with the 2316, and thus, it became convenient for users to install "lower case character ROM's" to get full upper and lower case on the video screen.

Keyboard Changes

The 1981 redesign of the keyboard system came about simply because the old "MM 5740 keyboard encoder" was no longer available. Apple switched to the AY-3600 (see Chapter 8) which is also used in the IIe and IIc. To accomodate the change, the new keyboard has a "two piece" design with a "piggy back" board (see Chapter 8, Figure 8.4).

This design has the fringe benefit of making it possible for just about any one to do the "Shift Key Mod" described in the caption for Chapter 8, Figure 8.4. To do the mod on the old one-piece keyboard requires soldering and a lot more expertise (see "Understanding the Apple II" by Jim Sather). A second fringe benefit of the use of the piggy back board was that it was convenient for Apple to design in a switch you can set to select whether your Apple II+ responds to Reset or only to "CTRL-Reset." A third benefit was the opportunity for Apple to change over to different style of sculpted keycaps and to do a few other upgrades to the "feel" of the keyboard (see Chapter 8).

Apple //e Revisions

The differences between the Apple //e and the Apple II+ are explained in detail in hundreds of places throughout the book, so this section just covers the various revisions of the IIe itself. The first 50-100,000 IIe's shipped have what is called the "Revision A" motherboard. Its most distinguishing characteristic for most folks is the inability to support super high res graphics. The part number for IIe motherboards is 820-0064-X wherein X is A, B, etc. It is written along the back behind the slots. Other differences between Rev. A and Rev. B, are a change to black writing on the keycaps instead of white, and disconnection of the built in "Shift Key Mod" (see Chap. 8) in the Rev. B.

Most of the half million or so //e's shipped in 1983 have the original set of //e ROMs. In late 1983, Apple began using a revised system ROM that enabled interrupts during scrolling of the characters in video display memory. This change was a response to a problem with 1200 baud modems (see Chap. 6 and Chap. 17). In the original ROM, scrolling is too slow, and interrupts are locked out, so incoming characters are lost at the end of every line.

Two more major ROM changes were completed in mid-1984, both of which help give IIc compatibility to the IIe. The first is a greatly improved interrupt handling system (see Chap. 9 and Chap. 27). Part of the change required use of addresses in the $C300 ROM and this can cause trouble for people who use a high resolution 80 column card in slot #3 rather than in the auxiliary slot. The other ROM change is in the character generator chip and it provides MouseText characters (see Chap. 26, fig. 11).

Apple //c

The first revision change for the Apple //c has to do with 1200 baud modems. In the original design for the //c, the baud clock for the 6551 ACIA serial communication chips is not quite correct (see Chap. 17). The buad clock signal is derived from the system clock and it is three percent too slow for the 6551's specifications. This can cause framing errors with some 1200 baud modems. The upgrade to the motherboard involved adding a special baud clock crystal for the 6551s.

Appendix E

Applications Guide to Expanding the Apple

Word Processing

Eye Strain:

High Resolution Characters

(When your eyes can distinguish the individual dots that make up a character, and when those dots are a little fuzzy around the edges, you launch into an endless round of unnecessary microfocusing. The ALS video board generates 9 by 11 characters as opposed to the //e's 5 by 7. The dots are so tiny that the characters appear to be solid. This places the burden of hard work on the video monitor and most of them on the market can't handle this resolution.)

- Videx "Ultraterm" (//e)
- ALS Smarterm I w/version 2.0 ROM

High Resolution Monitor

- Amdek "Video 300 A"
  (amber screen)
- Apple "Monitor II"
- Apple "Monitor III"

Typing Comfort:

Detachable Keyboard

- Zicor "Omega II"
  (all such keyboards are difficult to install)

Large Documents:

High Capacity Drives

- Microsci "A82"

Rana "Elite Two" or "Three"

(For CP/M or ProDOS software only, since DOS 3.3 software needs a Disk II compatible drive; for active editing, allow 128K of disk for 30 pages of text.)

Spontaneity:

Fast Co-processor

(The limits on your typing speed and on speed of reformatting the text on the screen come from the heart of the computing system.)

- Saturn "Accelerator"

(An Apple-on-a-card which runs 3.5 times faster than the standard Apple; for DOS 3.3, ProDOS, and Pascal word processors.)

- PCPI "Appli-Card" 6 MHz
  (Runs WordStar three times faster than standard CP/M cards.)

Speed for Global Editing:

RAMdisk Emulators

(Some editing operations such as search and replace, spelling checks, and indexing spend a lot of time looking through your text on the disk; a RAMdisk emulator eliminates the mechanical delays.)

- Axlon "Ramdisk 320"
  (battery backup)
- Legend 128K DE
- Saturn "Neptune" (192K for //e only)
- PCPI Ramextender
  (Working with PCPI Appli-Card, this is three to four times faster than any other.)

Keeping Track of your Drafts:

Time and Date Stamping

(It's easy to lose track of which version you wrote and when; ProDOS with Word Juggler can automatically record the time and date next to the file name in the disk directory.)

- Thunderware "Thunderclock"
- Practical Peripherals "ProClock"
  (Only ProDOS based word processors can do this, and these are the only clock cards that work with ProDOS.)

Printed Copy on Paper:

Letter Quality Printers

(To get proportional spacing, subscripts, superscripts, underlining, etc., you need an expensive printer, but not all printers work with all word processing programs. Buy the program before you buy the printer; each company sells more than one printer, and some word processors work with only one of the machines from a given manufacturer.)

- Diablo
- NEC
- Qume (Apple)
- Leading Edge (TEC) (C.Itoh)
  (Printing speeds range from 30 to 60 characters per second; you can get either a "parallel" or a "serial" printer, but your safest bet is to get a "Centronics Standard Parallel" printer and printer interface card.)

Teletype-style Printing

(If you don't care about the frills, you have two choices for less expensive printers: slow letter quality printers or fast dot matrix printers.)

- Comrex "CR-II"
  (12 characters per second, but the print looks like a good quality typewriter.)
- Epson FX-80
  (160 characters per second, but not very nice looking.)

Spreadsheets:

Large Models:

RAM Memory Expansion

(MagiCalc can use up to 512K of RAM in an Apple II and will automatically detect and adapt to nearly any RAM card. VisiCalc has a smaller maximum and will not work with all RAM cards.)

- Legend "S'Card"
  (This can carry from 128K on up to one megabyte of RAM.)
- Saturn "Neptune"
  (192K for //e only.)

Speed of Recalculation:

Fast Co-processors

(The bigger your model, the slower the recalculation; you can improve on number crunching speed by using a high speed version of your current processor.)

- Saturn "Accelerator"
  (3.5 times faster for all Apple based spreadsheets.)
- Microsoft "Premium Softcard //e"
  (For CP/M based SuperCalc.)
- PCPI "Appli-Card"
  (Three times faster for SuperCalc.)

More Information on the Screen:

Very High Resolution Video

(Apple standard is 40 columns, the //e and many "video cards" get you 80 columns, but with a very good monitor and video card, you can do even better.)

- Videx "Ultraterm"
  (Up to 160 columns; use Amdek 300A monitor.)

Comfortable Data Entry:

Numeric Keypads

(Keypads made for the II/II+ won't work in the //e, and //e keypads are much easier to install.)

- Trackhaus "//e Tender"
- ABT "numeric pad"
- Apple "numeric pad"
- CCP "Keywiz"

Database Management:

Large Databases:

Hard Disks

(Many DOS 3.3 database programs can't use hard disks, so you need a CP/M, MS-DOS 2.0 or ProDOS based program.)

- Apple "ProFile"
  (five megabytes)
- Apple 70 megabyte hard disk
- Mountain Dynamic hard disk
  (five, 10, or 15 megabytes)
- Davong hard disk

Speeding up the Sorting Process:

RAMdisks

(When sorting a large file, the speed bottleneck in programs such as dBase II is in the disk drive, so a fast processor won't help. A sort which takes half an hour on an Apple disk can be finished in 90 seconds with a PCPI RAMdisk.)

- Synetix "Flashcard"
  (294K)
- Axlon "RamDisk 320"
- Pion "Interstellar Drive"
  (Up to a megabyte with battery backup.)
- Legend "S'Card"
  (Up to a megabyte, no battery backup.)
- PCPI "RAM Extender"
  (128K or 256K; requires PCPI Applicard; three to four times faster than the others.)

Software Compatibility:

Co-processors

(In business, the great majority of microcomputer database work is done with dBase II or Lotus 1-2-3, and these will not run on an unenhanced Apple.)

- Microsoft "SoftCard"
- PCPI "Appli-Card"
  (CP/M for dBase II.)
- Rana "8086/2"
  (Program, disk, and video compatible with IBM PC for dBase II or Lotus on MS-DOS 2.0.)

Mainframe Computer Access:

Communication by modem

(The slow but cost efficient way to exchange data with a mainframe is via a telephone modem. To make your work more convenient, you need "terminal emulation" software such as "Softerm" from Softronics, Inc. which can make the Apple act like an IBM 3101, DEC VT52, DEC VT100, Data Gen. D200, Lear Siegler ADM 3A or 5, Hazeltine 1400 or 1500, ADDS Reagent, or Televideo 900 series terminal. "ASCII Professional" from Southwestern Data Systems or "TermExec" from Exec Software let you customize for special needs.)

- Hayes "Smartmodem 1200"
  (Needs serial card for interface.)
- Hayes Micromodem
  (300 baud, doesn't need interface card.)
- Microcom ERA2
  (1200 baud modem card w/terminal emulation.)

3270 Terminal Emulation

(IBM 3270 terminals connect with IBM mainframes via a high speed coaxial cable link. You need software emulation of the 3270 terminal and a sophisticated hardware interface.)

- Apple "IBM 3270 Cluster Controller"
  (for four or seven Apples of any model.)
- AppleLine
  (Connects Apple to existing cluster controller from another manufacturer.)
- Protocol Card
  (Handles terminal emulation of 3270.)

Nine Track Mag Tape

(The Apple can be interfaced directly to an industry standard mag tape drive for bulk transfer of large amounts of data.)

- IDT "TD 1012"
  (1600 bpi, PE.)
- Electrovalue
  (800 bpi, NRZI)

Appendix F

Listing by Manufacturer

This appendix has been provided as a general information guide. The products listed and the prices are subject to change. Due to the constant fluctuations in the industry neither the prices nor the products listed here can be guaranteed for accuracy. Where no price was available at the time of publishing, an N/A symbol has been substituted.

ALF
1315F Nelson St., Denver, CO 80215
(303) 234-0871

– MC1 Music Card nine voice music synthesizer	195.00
– MC16 Music Card three voice music synthesizer	195.00
– DC3 Apple/IBM disk controller	350.00
– AD8088 Processor card with FTL and MET	345.00
– Expansion Board for AD8088 (sockets for 64/128K)	295.00
– 64K	75.00
– 8087	262.50
– ADGS high speed video plotting system	N/A
– HGR6 //e super Hi Res software	50.00

Abacus
P.O. Box 1836, Detroit, MI 48231
(800) 835-2246 (313) 524-2444

– Know-Drive w/Playback	485.00

Abacus Compusource
510 N. First Ave., Suite 408, Minneapolis, MN 55403
(612) 340-1468

– Portable Apple compatible computer	1,795.00
– PC Mate upgrade for PC compatibility	N/A

Acorn
4455 Torrance Blvd. #108, Torrance, CA 90503
(213) 371-6307

– 68000 Board	1,500.00

Advanced Business Technology
12333 Saratoga-Sunnyvale Road, Saratoga, CA 95070
(408)446-2013

– Keypad 13 numeric keys	125.00
– Keypad 13 numeric keys for //e	145.00
– Softkey 15 function keys	150.00
– Symbtrak Bar Code Reader and Printer	500.00
– Retail Manager	2,200.00
– Bar Code Wand	225.00

Advanced Color Technology
21 Alpha Rd., Chelmsford, MA 01824
(617) 256-1222

– ACT-1 Ink Jet Color Printer	N/A
– ACT-2 Ink Jet Color Printer	6,150.00
– direct composite video interface	2,345.00
– serial interface	260.00

Advanced Logic Systems
2685 Marine Way, Suite 1209, Mountain View, CA 94043
(415) 594-1671 (800) 235-6442

– Smarterm II	180.00
– Dirt Cheap Video, 64 col. w/RF mod. for TV	90.00
– Color II (RGB Interface)	180.00
– Dispatcher	140.00
– Printermate	100.00
– Smarterm I	345.00
– Version 2.0 ROM	40.00
– 16K RAM card	100.00
– Z-card II	170.00
– CP/M Card Z-80B	400.00

Adwar
335 W.35th St., New York, NY 10001
(212) 947-4040

– ARS-170A (Apple-NTSC graphics converter)	1,495.00
– ARS-170AX with genlock	1,995.00
– Apple Proc Mod	100.00

Alien Group

– Voice Box	215.00

Amdek
2201 Lively Blvd., Elk Grove Village, IL 60007
(312) 364-1180

– DVM II RGB board	180.00
– DVM 80e RGB board	195.00
– DVM III multiple mode RGB board	195.00
– Amdek Video-300A amber monitor	200.00
– Amdek Video 300 green monitor	180.00
– Color I Lo Res	380.00
– Color I+ Lo Res	400.00

– Color II (= II+) 560 by 240, digital color, med res	560.00
– Color III Lo Res	450.00
– Color III+ md res	480.00
– Color IV 720 by 420, 16 MHz, analog color range	690.00
– Amplot-II digital plotter, 6 pen	1,099.00
– DXY-100 plotter	750.00
– Amdisk-1 3 inch drive 286K	300.00

Amkey
220 Ballardvale St., Wilmington, MA 01887
(617) 658-7800

– PRO 100 keyboard	295.00

Analytical Engines
3415 Greystone, Suite 305, Austin, TX 78731
(512) 346-8430

– Saybrook 68000, 8 MHz	900.00
– Saybrook 68000, 12.5 MHz	1,200.00
– Saybrook 68000, 14 MHz	1,400.00

Anchor Automation
6913 Valjean Ave., Van Nuys, CA 91406
(818) 997-6493

– Signalman Mark I 300 baud	100.00
– Volksmodem (battery powered)	70.00
– Signalman Mark VII autodial/auto answer 300 baud	160.00
– Signalman Mark XII autodial/auto answer 1200 bps	400.00

Androbot, Inc.
P.O. Box 9-214, San Jose, CA 95103
(408)262-4914

– Topo Robot	1,595.00

Anthro-Digital
103 Bartlett Ave., Pittsfield, MA 01201
(413)448-8278

– Amper-Magic Ampersand Commands	75.00
– Amper-Magic Command Library	35.00

A +
P.O. Box 2965, Boulder, CO 80321
(212)725-4694

– Magazine; one year subscription	25.00

Apple Computer
20525 Mariani Ave., Cupertino, CA 95014
(408) 973-2222 (800) 538-9696

– Monitor II	230.00

– Monitor III	260.00
– //e Text Card	125.00
– //e 64K Extended Text Card	295.00
– RF modulator	N/A
– Graphics tablet	800.00
– Numeric key pad	160.00
– Applemouse	150.00
– Joystick	60.00
– Paddle	29.00
– Hobby/Prototyping Board	N/A
– Parallel Interface	165.00
– Communications Board	N/A
– Super Serial	225.00
– Serial Interface Card	N/A
– IEEE-488	450.00
– 3270 Cluster Controller, 3 ports	4,500.00
– 3270 Cluster Controller, 7 ports	7,000.00
– Apple Line connect to standard 3270 controller	1,295.00
– Apple Modem 300	225.00
– Apple Modem 300/1200	495.00
– Protocol Card for 3270 emulation	700.00
– Letter Quality Printer	2,195.00
– Imagewriter	595.00
– Laser printer	5,000.00
– Silentype	350.00
– Color Plotter Model 410, 4 pen	995.00
– Language Card	100.00
– ROM CARD	N/A
– Disk II	395.00
– Duodisk (2 by thinline 140K)	800.00
– 3½ inch drive 400K	495.00
– ProFile, 5 megabyte	1,500.00
– 70 megabyte hard disk	7,000.00

Apple Orchard
P.O. Box 6502, Cupertino, CA 95015
(408)727-7652

– Magazine; one year subscription	24.00

Applied Analytics
8235 Penn-Randall Pl., Upper Marlboro, MD 20772
(301) 420-0700

– Microspeed II (2 MHz AMD 9511)	495.00
– Microspeed II+ (4 Mhz AMD 9511)	645.00
– Booster 4 MHz 6502C	600.00

Applied Engineering
P.O. Box 470301 Dallas, TX 75247
(214) 492-2027

– Viewmaster 80 (7 x 9)	170.00

– Z-80 Plus	140.00
– Super Music Synthesizer sixteen voice	160.00
– Timemaster	130.00
– A/D converter, 8 bit, 8 channel	130.00
– A/D converter, 12 bit, 16 channel	320.00
– Serial input/output	62.00
– Memory Master //e (64K)	170.00
– Memory Master //e (128K)	250.00

Artra, Inc.
P.O. Box 653, Arlington, VA 22216
(703) 527-0455

– Waldo Voice Activated Control System	600.00
– w/"robot voice" unlimited vocabulary	+199.00
– w/"human quality voice" 206 words	+199.00
– BSR direct connect	70.00

Axiom (Seikosha)
1014 Griswold Ave., San Fernando, CA 91340
(213) 365-9521

– EX-1620 240-960 cps	795.00
– TX-1000 B & W video printer (accepts video signal)	3,395.00
– IMP-40 100 cps	N/A
– GP-100A, 30 cps	300.00
– GP-250X 50 cps dot matrix	340.00
– color printer	600.00

Axlon
70 Daggett Drive, San Jose, CA 95134
(408) 747-1900

– Ramdisk 320 (external, battery backup)	1,000.00
– Ramdisk 128 card	380.00
– 5MB Hard Disk	1,200.00
– 5MB Removable Hard Disk w/5MB fixed HD	1,900.00

BMC
16830 South Avalon Blvd., Carson, CA 90746
(213) 515-6005

– BM12 AU Monochrome monitor	130.00

Bausch and Lomb (Houston Instruments)
8500 Cameron Rd., Austin, TX 78753
(512) 835-0900 (800) 531-5205

– Hiplot DMP-29, 8 pen, 11 by 17	2,295.00
– Hiplot DMP-40	995.00
– DMP-41	3,000.00
– D-5000 Omniscribe analog strip chart recorder	1,000.00
– 4500 Microscribe smart analog strip chart	835.00
– Complot CPS-19	14,000.00

– Complot CPS-20	4,000.00
– Complot CPS-30	6,000.00
– HiPad DT-114 digitizer	N/A

Beagle Brothers
7th Floor, 4315 Sierra Vista, San Diego, CA 92103
(800)227-3800 ext. 1607

– GPLE w/DOS Mover and PEEKs and POKEs chart	50.00

Bi Comm Systems
10 Yorkton Industrial Court, St. Paul, MN 55117
(612) 481-0775

– PC-1 power line controller	265.00

C.Itoh (see Leading Edge)

California Computer Systems
250 Caribbean, Sunnyvale, CA 94086
(408) 734-5811

– Arithmetic Processor (AMD9511)	360.00
– Wire Wrap Board (out of production)	N/A
– Soldertail Board (out of production)	N/A
– Extender Board (out of production)	N/A
– Etch Board (out of production)	N/A
– Model 7424 Calendar/Clock Module	115.00
– Modle 7470 Analog to Digital Converter	105.00
– Model 7440 Programmable Timer	120.00
– Model 7490 IEEE-488 Interface	200.00
– Model 7710A Asynchronous Serial (for printer)	135.00
– Model 7710D Asynchronous Serial (for modem)	135.00
– Model 7711 Asynchronous Serial (modem/printer)	110.00
– Model 7712 Synchronous Serial Interface	175.00
– Model 7720 Parallel	100.00
– Model 7728 Centronics Parallel Interface (out of prod.)	N/A
– Model 7731 Centronics Parallel Interface	70.00
– Model 7114 12K ROM/PROM Module	115.00

Call A.P.P.L.E.
21246 68th Ave. S., Kent WA 98032
(206)872-2245

– *All About Applesoft* w/diskette	24.00
– Magazine and club membership	51.00

Canon USA, Inc., Systems Div.
One Canon Place, Lake Success, NY 11042
(516)488-6700

– PJ-1080A Ink Jet Color Printer	800.00
– LBP-CX Laser Printer	3,000.00

Chalkboard
3772 Pleasantdale Rd., Atlanta, GA 30340
(800) 241-3989 (404) 496-0101

– Power Pad	100.00

Chatsworth Data Corporation
20710 Lassen St., Chatsworth, CA 91311
(213) 341- 9200

– 500 Series Hand Fed Card reader	1,095.00
– 2000 Series Card Reader	1,675.00

Coex (Components Express)
E. Edinger Ave., Santa Ana, CA 92705

– 16K RAM	100.00
– Parallel Card	100.00
– Protocard	30.00
– Extender Card	30.00
– //e Extended RAM Card	200.00

Colne Robotics
207 N.E. 33rd St., Ft. Lauderdale, FL 33334
(305)566-3101

– Armdroid 1	N/A

ComX

– //e 64K, 80 Col.	300.00
– 16K Ramcard II+	180.00

Computer Stations
11610 Page Service Dr., St. Louis, MO 63146
(314) 432-7120 (800) 325-4019

– Dithertizer (camera, dithertizer, printer interface)	1,175.00

CTA (Computer Technology Associates)
1704 Moon, N.E., Suite 14, Albuquerque, NM 87112
(505) 298-0942

– Apple Prom EPROM Burner	150.00
– Generic Serial Communication Card	110.00
– Touch Screen Bezel	700.00

Comrex International, Inc.
3701 Skypark Drive, Suite 120, Torrance, CA 90505
(213) 373-0280

– CR-5400 20MHz, 800 line (P31, P39, or PDB - amber)	100.00
– CR-5600 20MHz, 1,000 line (P31, P39)	135.00
– CR-5600 20 MHz, 1,000 line amber	150.00
– CR-6500 RGB 300 by 260	325.00

– ComRiter CR-I, 17 cps (Qume Sprint 3)	800.00
– ComRiter CR-II, 12 cps	600.00
– ComRiter CR-III, 23 cps	1,000.00
– ComDrive CR-1000, dual thinline	600.00
– ComScriber I plotter	795.00

Corona Data Systems
31324 Via Colinas, Westlake Village, CA 91361
(213) 991-1144

– Starfire Hard Disk 5 Megabyte	2,195.00
– 10 Megabyte	2,695.00

Corvus
2029 O'Toole Ave., San Jose, CA 95131
(408) 946-7700

– Interface card	300.00
– Hard Disk 6 AP	2,095.00
– Hard Disk 11 AP	2,750.00
– 20 Meg.	3,750.00
– The Bank 200 Megabyte backup tape	2,200.00

Cramapple Adapter
Box 98, Cambridge B Branch, Boston, MA 02140
(617) 628-0206

– (remove motherboard 48K, install 192K)	
– no-mod adapter	120.00
– adapter requiring motherboard modification	80.00
– 24 64K bit RAM chips for 192K total	250.00
– 256K bit RAM chip adapter for //e	120.00

Creative Computer Peripherals, Inc.
1044 Lacey Road, Forked River, NJ 08731
(800) 225-0091 (609) 693-0002

– Keywiz 83	300.00
– Keywiz Convertible	300.00
– Keywiz VIP	440.00

Cyborg
342 Western Ave., Boston, MA 02135
(617) 782-9820

– ISAAC	
– 41A chassis	1,600.00
– 91A system chassis	4,000.00
– 850-130 2 by I slot chassis	150.00
– 850-129 6 by I slot chassis	700.00
– 2000 system chassis	7,925.00
– 850-161 6 by C slot chassis	275.00
– I-140A analog input conditioner	800.00
– I-140B analog input conditioner	750.00

– I-140C analog input conditioner	750.00
– I-100 A/D converter	850.00
– I-150 A/D converter	1,100.00
– I-180 A/D converter	700.00
– I-130 A/D converter	600.00
– C-100 A/D converter	3,825.00
– C-200 256K RAM	2,100.00
– I-110 analog output	750.00
– I-120 digital I/O	350.00

DOSS Industries
1224 Mariposa, San Francisco, CA 94107
(415) 861-2223

– Apple Center, case w/fan and surge protector	240.00

DTC (Data Terminals and Communication)
590 Division St., Campbell, CA 95008
(408) 378-1112

– 380Z, 32 cps (Diablo code)	1,495.00

Dan Paymar
91 Pioneer Pl., Durango, CO 81301
(303) 259-3598

– Lowercase Adapter	37.50
– Lowercase Adapter-Rev. 7	27.50

Dark Star Systems
P.O. Box 140, Dept. N., Amherst. MA 01004
(413) 584-7600

– Snapshot //e	140.00
– Snapshot II	120.00

Data Cue, Inc.
5696 Highway 431 South, Brownsboro, AL 35741
(205) 883-2933

– Disc Master II, 8 inch or 3 1/2 inch MFM controller	265.00

Data Translations
100 Locke Drive, Marlboro, MA 01752
(617) 4811-3700

– DT2832 A/D Converter	695.00
– 5714	+550.00
– 5716	+975.00
– DT2834 A/D Converter	795.00
– DT710 screw terminal panel	190.00

Davong
217 Humboldt Court, Sunnyvale, CA 94086
(408) 734-4900

– Hard Disk	
– 5 Megabyte	1,995.00
– 10 Megabyte	2,400.00
– 15 Megabyte	2,800.00
– 21 Megabyte	3,300.00
– 32 Megabyte	4,000.00
– Tape Backup	1,995.00
– Network	N/A

Decillionix
P.O. Box 70985, Sunnyvale, CA 94086
(408) 735-0410

– DX-1 Sound Processing System	240.00

Diablo
P.O. Box 5030, Fremont, CA 94537
(800)824-7888 (415) 786-5000

– Model 630, 40 cps	1,995.00
– Model 620, 20 cps	1,050.00
– Series C Ink Jet Printer, 20 cps	1,295.00

Digital Acoustics
1415 E. McFadden, Ste. F, Santa Ana, CA 92705
(714) 835-4884

– DTACK Grande (68000)	
– 128K	795.00
– 1 megabyte	1,995.00
– DTACK Grounded (68000)	
– 4K high speed RAM	683.00
– 92K	1,123.00

Digital Electronics Systems
107 Euclid Ave., Mountain Brook, AL 35213
(205) 871-0987

– Removable Hard Disk	1,300.00

Digital Research
P.O. Box 579, Pacific Grove, CA 93950
(800) 227-1617 ext. 400 (800)772-3545 ext. 400

– Gold Card Z-80B for CP/M 3.0 w/64K RAM	495.00
– 128K RAM add-on board	325.00
– 192K Gold Card	775.00

Don't Ask
2265 Westwood Bl., Suite. B-150, Los Angeles, CA 90064
(213) 477-4514

– S.A.M. Software Automatic Mouth (includes D/A converter and amplifier)	150.00

Douglas Electronics
718 Marina Blvd., San Leandro, CA 94577
(415) 483-8770

– Appleseed motherboard BX DE-12	95.00
– Extender Board 6 DE-12	18.00
– General Purpose Bread Board 25 DE-12	24.00
– Wire Wrap Panel 32 DE-12	21.00

E-mu Systems
2815 Chanticleer, Santa Cruz, CA 95026

– Drumulator	995.00

East Side Software
344 E. 63 St., Suite 14-A, New York, NY 10021
(212) 355-2860

– Wildcard	110.00
– Wildcard 2 for 64K or 128K //e	140.00
– Wildcard Plus (has 6502 on board)	170.00

Eco-Tech, Inc.
2990 Lake Lansing Rd., East Lansing, MI 48823
(517) 337-9226

– ALIS A12 Precision Analog input	1,517.00
– ALIS A08 Analog input	1,149.00
– ALIS AO Anaolog output	841.00
– ALIS DIO Digital input/output	1,600.00

Eicon Research Inc.
Viking Way, Bar Hill, Cambridge, CB5 8EL, England, U.K.
2157 Park Boulevard, Box 60456, Palo Alto, CA 94306
(415) 326-2146

– DisCache Hard Disk (w/RAM for high speed anticpatory track buffering, network server, incremental backup)	
– 10 Megabyte w/128K buffer	3,350.00
– 20 Megabyte w/256K buffer	4,250.00
– 40 Megabyte	4,900.00
– Tera-Drive 2 megs in two thinline 5 1/4 inch floppies	N/A

Einstein Corp.
11340 W. Olympic Blvd., Los Angeles, CA 90064
(213)477-4539

– Expediter II compiler	150.00

Electronic Data Protection
P.O. Box 673, Waltham, MA 02254
(617) 891-6602 (800) 343-1813

– Lemon (EC-I)	60.00
– Lime (EC-II)	90.00

– Peach (EC-IV)	100.00
– Orange (EC-V)	140.00
– Kiwi	N/A
– Hawk power monitor	200.00
– Groundhog anti-static mat	90.00
– Grizzly 200 watt	900.00
– Grizzly 500 watt	2,400.00
– Grizzly 1,000 watt	5,200.00

Electrovalue Industrial, Inc.
P.O. Box 157N, Morris Plains, NJ 07950
(201) 267-1117

– Model 1025 full size mag tape, 800 bpi	3,000.00
– Pertec 7000 7 inch tape	1,800.00

En-Link, Inc.
4706 Bond, Shawnee, KS 66203
(913)268-6066

– Ethernet Interface	N/A

ETC (Enhancement Technologies)
Box 1267, Pittsfield, MA 01202
(413) 445-4219

– PDQ II (68000) board	1,495.00

Enter Computer, Inc.
6867 Nancy Ridge Drive, San Diego, CA 92121
(800)421-5426 (619) 450-0601

– Sweet-P plotter	800.00
– Six-Shooter, 6 pen, 11 by 17	1,095.00

Epson
3415 Kashiwa St., Torrance, CA 90505
(213) 539-9140 (800) 421-5426

– MX-80	525.00
– MX-100	750.00
– RX-80	500.00
– FX-80	700.00
– FX-100	900.00

Excalibur Technologies
1250 Oakmead Parkway, Suite 210, Sunnyvale, CA 94086
(408) 773-1550

– Savvy Language System	
– Savvy One	350.00
– Savvy Pro	500.00
– Business Savvy	950.00

EPS (Executive Peripherals)
800 San Antonio Road, Palo Alto, CA 94303
(415) 856-2822

– Detachable keyboard	400.00

FM Panatronics
Box 1088, Stone Mountain, GA 30086

– Programmable Serial Card	150.00

FMJ
Box 5281, Torrance, CA 90510
(213) 325-1900

– Sentry II Cool Stack	175.00

Fender/Rogers/Rhodes
1300 Valencia Dr., Fullerton, CA 92631
(714) 879-8080

– Rhodes Chroma Music Terminal, 16 voice	5,295.00
– expansion module for additional 16 voices	3,150.00
– Apple interface kit	495.00

Gibson Laboratories, Inc.
23192-D Verdugo Dr., Laguna Hills, CA 92653
(619)730-3088

– Light Pen LPS II	350.00

Hayden Software
600 Suffolk St., Lowell, MA 01853
(800)343-1218 (617)937-0200

– ORCA/M assembler	150.00
– Applesoft compiler	60.00

Hayes Microcomputer Products
5923 Peachtree Industrial Blvd., Norcross, GA 30092
(404) 449-8791

– Chronograph	250.00
– Micromodem II	380.00
– Micromodem //e	290.00
– Smartmodem 300	290.00
– Smartmodem 1200	700.00

Hayes Products
1558 Osage St., San Marcos, CA 92069
(619) 744-8546

– Mach II Joystick	40.00
– Mach III Joystick w/fire button	50.00

Hewlett-Packard
16399 W. Bernardo Drive, San Diego, CA 92127
(619) 560-9414

– 7470 2 pen plotter	1,095.00
– 7475 6 pen plotter	1,895.00
– ThinkJet ink jet printer	495.00

High Order Microelectronics
17 River Street, Chagrin Falls, OH 44022
(216) 247-3110

– Repeaterrr	27.00
– Repeaterrr +	37.00

Hollister Microsystems
1455 Airport Blvd., San Jose, CA 95110
(408) 293-3900

– HMS 3264 EPROM Programmer	395.00

Hollywood Hardware
6842 Valjean Avenue, Van Nuys, CA 91406
(818) 989-1204

– PRO-1 prototype board	30.00
– AD 121602	300.00
– PD48 Buffered digital interface, 48 lines	250.00
– Ultra-ROM Board	190.00
– A16G op amp unit	80.00

Houston Instruments (see Bausch and Lomb)

IBS Computertechnik
1011 Rose Marie Lane 16, Stockton, CA 95207
(209) 473-7473

– AP20 Intemex w/68000 CPU and 128K	650.00
– 256K RAM for AP20	720.00

In Cider
P.O. Box 911, Farmingdale, NY 11737
(603) 924-9471

– magazine; one year subscription	25.00

Innovative Data Technology
4060 Morena Blvd., San Diego, CA 92117
(619) 270-3990

– IDT-TD 1050 system	8,500.00

Innovative Micro Goodies
34732 Calle Fortuna, Capistrano Beach, CA 92624
(714) 661-0435

– D-Tach keyboard carrier	90.00

Integral Data Systems
Millford, N.H. 03055
(603) 673-9100 (800) 258-1386

– Prism 480, 100 cps	650.00
– P-Series 80	1,300.00
– P-Series 132	1,500.00
– color	+400.00

Interactive Microware
P.O. Box 771, Dept. 3, State College, PA 16801
(814) 238-8294

– Adalab	495.00
– Ada-Mux	195.00
– Ada-Byte (32 bit digital multiplexer)	395.00
– Ada-Amp w/prog. gain and multiplexer	300.00
– Chromadapt	350.00

Interactive Structures
146 Montgomery Avenue, Bala Cynwyd, PA 19004
(215) 667-1713

– Pkaso EP12-80		175.00
– Pkaso EP12-100		175.00
– Pipeline 64K buffer		230.00
– Shuffle Buffer		300.00
– DAISI (Data Acquisition and Instrumentation Systems Interface		
– AI13 (12 bit analog input)		550.00
– AI02 (8 bit analog input)		299.00
– AO03 (8 bit analog output)	2 channel	195.00
	4 channel	275.00
	8 channel	440.00
– DI09 (Digital timer interface)		330.00
– SC-14 Signal conditioners	1 channel	44.00
	4 channel	275.00
	16 channel	440.00
– UI-16 PB power control	4 channel	95.00
	8 channel	120.00
	16 channel	155.00
– UI-16 PAM power control		395.00

Juki Industries of America, Inc.
20437 South Western Ave., Torrance, CA 90501
(213) 320-9001

– Model 6100 daisy wheel printer, 18 cps	700.00

Keithley/DAS
349 Congress St., Boston, MA 02210
(617) 423-7691

– Chassis w/software	2,100.00
– AIM1 input signal conditioner	850.00

– AIM2 input signal conditioner	300.00
– AIM3 input signal conditioner	575.00
– AIM4 input signal conditioner	760.00
– AIM5 input signal conditioner	780.00
– AIM6 input signal conditioner	680.00
– ADM1 A/D Converter	700.00
– ADM2 A/D Converter	950.00
– AOM1 Analog Output, 2 channel	500.00
– AOM1 Analog Output, 5 channel	800.00
– AOM2 Analog Output, 1 channel	700.00
– AOM2 Analog Output, 2 channel	1,400.00
– AOM3 Analog	N/A
– DIM1 digital input	250.00
– DOM1 digital output	250.00
– PCM1 power control	350.00
– PCM2 power control	500.00

Kemcore Company
Suite 7068, 111 East Drake, Fort Collins, CO 80525

– Fan and back panel	155.00

Kensington Microware
919 Third Avenue, New York, NY 10022
(212) 486-7707

– System Saver Fan	90.00

Key Tronic
P.O. Box 14687, Spokane, WA 99214
(800) 262-6006

– KB-200	300.00

Keytec
P.O. Box 722, Marblehead, MA 01945
(617) 292-6484

– keyboard enclosure	60.00

Koala Technologies Corp.
3100 Patrick Henry Dr., Santa Clara, CA 95050
(800) 227-6703 (408) 986-8866

– Touch Pad	125.00

Kraft
450 W. California Ave., Vista, CA 92083
(619) 724-7146

– Premium Joystick	65.00
– Premium Game Paddle Pair	50.00

Lazer Systems
925 Lorna St. Corona, CA 91720
(714) 735-1041

– Lowercase Plus	60.00
– Keyboard Plus	30.00

Leading Edge (TEC) (C.Itoh)
225 Turnpike St., Canton, MA 02021
(617) 828-8150 (800) 343-6833

– Prowriter 8510A, 120 cps	595.00
– Prowriter 2 Printer-1550, wide carriage, 120 cps	995.00
– Starwriter F10-40	1,895.00
– Printmaster F10-55	2,000.00
– Gorilla (Seikosha GP-100A) 30 cps	250.00

Legend Industries
2220 Scott Lake Rd., Pontiac, MA 48054
(313) 674-0953

– Slot 8	N/A
– 18K SRC battery backup	160.00
– 64K KC	327.00
– 128K DE	600.00
– S'Card 64K to 1 Megabyte (w/256K chips)	N/A
– 64K	400.00
– 128K	525.00
– 192K	624.00
– 256K	725.00

Lobo
358 S. Fairview Ave., Dept. A0583, Goleta, CA 93117
(800) 235-1245 (800) 322-6103 (Inside CA)

– Apple Compatible Drive	385.00
– 8 inch Drives	1,925.00
– 5 Megabyte Hard Disk w/floppy	2,675.00

M & R Enterprises
910 George Street, Santa Clara, CA 95050
(408) 980-0160

– Sup'R Mod RF Modulator	70.00
– Sup'R Terminal	350.00
– Sup'R Switcher	295.00
– Sup'R Fan	50.00
– Adaptabox	50.00
– Sup'R Access-1	700.00

MBI
1019 8th St., Golden, CO 80401
(303) 279-8438

– VIP Graphics Card w/serial port	120.00
– Appletime Clock Card	85.00

MPC Peripherals
9424 Chesapeake Dr., San Diego, CA 92123
(619) 278-0630

– 16K board	149.00
– AP-SIO, serial card, includes Imagewriter driver	169.50
– AP-96 Lowercase generator	29.50
– AP-80 Parallel Printer Card	130.00
– Omnibuffer	400.00
– Bubdisk 128K bubble memory	875.00
– Ramtext //e (64K)	290.00
– Graphics parallel	175.00
– 64K Buffer card	300.00
– Graphics parallel w/64K buffer	370.00

Macrotech Computer Products Ltd.
1370 Marine Drive, North Vancouver, B.C., V7P 1T4
(604) 984-9305
USA: 1400B Kentucky St., Bellingham, WA 98228

– Macromem-1 16K	100.00
– Macromem-2 32K	180.00
– Macromem-3 64K	300.00
128K	450.00
– Diskulator	
– 64K	450.00
– 256K	550.00
– 512K	1,100.00
– Macroprint parallel card w/zoom	180.00

Magellan
4371 East 82nd St., Suite D, Indianopolis, IN 46250
(317) 842-9138

– Light Pen	190.00

Mannesman Tally
8301 South 180th St., Kent, WA 98032
(206) 251-5524 (800) 447-4700

– MT 160L, 160 cps	770.00
– Spirit	400.00

Metamorphic Systems Inc.
8950 Ville La Jolla Dr., Suite 1200
(619) 457-3870 (800) 228-8088

– MetaCard	
– 64K	850.00
– 128K	980.00

Micro-Analyst Inc.
P.O. Box 15003, Austin, TX 78761
(512) 926-4527

– Replay II	80.00

Micro Computer Technologies
1745 21st St., Santa Monica, CA 90404

– SpeeDemon	295.00

Micro Control Systems
230 Hartford Turnpike, Vernon, CT 06066
(203) 872-0602

– Space Tablet 3-axis digitizer	475.00

Micro Dimensions
4860 East 345th St., Willoughby, OH 44094
(216) 953-8414

– MicroD-1000 lab interface	300.00

Micro Display Systems
P.O. Box 455, Hastings, MN 55033
(612) 437-2233

– The Genius 57 line display	1,795.00

Micro General Corporation
1929 S.E. Main St., Irvine, CA 92714
(714) 557-3744

– Weighmate (scale)	695.00

Micro Mint
561 Willow Ave., Cedarhurst, NY 11516
(516) 374-6793 (800) 645-3479

– Sweet Talker	100.00
– Micromouth	150.00
– MicroVox	300.00

Micro Sci
2158 South Hathaway St., Santa Ana, CA 92705
(714) 241-5600

– A 40 Disk Drive (160K)	380.00
– XL-drive (143K)	199.00
– A 2 Disk Drive (143K)	345.00
– A 70 Disk Drive (280K)	530.00
– A 82 Disk Drive (320K)	570.00
– Disk Controller	100.00
– 80/64e //e 80 colum card	170.00
– HAVAC slotless Apple compatible computer w/drive	850.00

Micro Works
P.O. Box 1110 Del Mar, CA 92014
(619) 942-2400

– DS-65 Digisector digitizing TV camera	600.00
digisector board only	350.00

Microcom, Inc.
1400A Providence Highway, Norwood, MA 02062
(800)322-ERA2 (617)762-9310

– ERA2 1200 baud modem card w/terminal emulation	430.00

Micrographic Images
19612 Kingsbury St., Chatsworth, CA 91311
(818) 368-3482

– NEC 7220 board	N/A
– Video digitizer	N/A

Micromax
6868 Nancy Ridge Dr., San Diego, CA 92121
(619) 457-3131

– Viewmax-80	230.00
– Viewmax-80e (64K)	190.00
– Viewmax-80e (128K)	300.00
– Ultimax	N/A
– PrintMax	90.00
– GraphMax	150.00
– GraphMax w/color and zoom	170.00
– Z-Max, 4 MHz Z-80A	160.00

Microsoft
10700 Northup Way, Bellevue, WA 98004
(206) 828-8080

– 16K card	100.00
– SoftCard	345.00
– SoftCard //e	345.00
– Premium SoftCard //e Z-80B	495.00
– ALDS assembler	125.00
– TASC compiler	175.00

MicroSparc Inc.
10 Lewis Street, Lincoln, MA 01773
(617) 259-9710

– Ampersoft; ampersand package	50.00

Microtek
4750 Viewridge, San Diego, CA 92123
(800) 854-1081 (619) 569-0900

– Magnum 80e	50.00
– Magnum 80	200.00
– Magnum 80Me (64K)	140.00
– Rainbow 16A	120.00
– Rainbow 256 (analog RGB board)	150.00
– RV-611C Parallel Interface 7 or 8 bit parallel	70.00
– Dumpling-GX (2K buffer)	90.00
– Dumpling-16	215.00
– Dumpling-32	240.00
– Dumpling-64	285.00
– Bufferbox 64K	400.00
– 64K add on kit	100.00
– SV-622 Serial Interface Card	100.00

– Bam-16, 16K card	100.00
– Bam-64	200.00
– BAM 128K DE	300.00
– Q-Disk, 128K RAM	400.00

Mimco
1547 Cunard Rd., Columbus, OH 43227
(614) 237-3380 (800) 454-3801

– Joystick	60.00

Mitac
P.O. Box D, Santa Clara, CA 95055
(408) 988-0258

– Mate-I Apple Compatible	170.00
– 10 MB Hard Disk	1,500.00
– Monitor	130.00

Mountain Computer
300 El Pueblo Rd., Scotts Valley, CA 95066
(408) 438-6650

– Model 1100 A Card Reader	1,500.00
– Music System	395.00
– Supertalker SD 200	200.00
– Expansion Chassis	750.00
– A/D + D/A	350.00
– CPS Multifunction (out of production)	240.00
– The Clock	280.00
– Ramplus+ (32K)	220.00
– Romplus+	155.00
– ROM Writer	175.00
– Dynamic Hard Disk System	N/A
– 5 Megabyte	1,995.00
– 10 Megabyte	2,495.00
– 15 Megabyte	2,995.00
– 20 Megabyte	3,495.00
– FileSafe Tape Backup System (20 Megabyte)	2,195.00
– Diskette quantity duplications systems	N/A
– 3 ½ inch	N/A
– 5 ¼ inch	N/A
– 8 inch	N/A

Multi-Tech Systems Inc.
82 Second Avenue S. E., New Brighton, MN 55112
(612) 631-3550

– Modem II	400.00
– Multi-Modem II (300/1200)	N/A
– MT212 AD	700.00
– MT103	N/A
– FM30 Acoustic couplers	N/A
– MT202T 1200 bps	N/A

– MT25 break out/monitor	N/A
– MT1001D Direct Connect data coupler	N/A
– Mulit-Mux 402	N/A

Multitech Electronics
195 West El Camino Real, Sunnyvale, CA 94086
(800) 538-1542 (408) 773-8400

– SSB-Apple speech synthesizer	200.00
– Accu-Feel MAK-II keyboard	150.00

NEC Home Electronics
1401 Estes Ave., Elk Grove, IL 60007
(312) 228-5900

– 8923A, dot matrix printer, 100 cps	645.00
– 8025A, dot matrix printer, 120 cps	975.00
– 15LQ, letter quality printer	700.00
– 1201 Hi Res Green Monitor	285.00
– 1205 Hi Res Amber Monitor	210.00
– 1212 Composite, Low Res color	400.00
– 1216 RGB Hi Res color, 640 by 240, 10 MHz	600.00

NEC Information Systems
5 Militia Dr., Lexington, MA 02173
(617) 862-3120 (800) 343-4419

– Series 2000 Letter quality printers, 20 cps	1,100.00
– Series 3500 Letter quality printers, 33 cps	2,200.00
– Series 7700 Letter quality printers, 55 cps	2,600.00

Nestar Systems
2585 E. Bayshore Rd., Palo Alto, CA 94303
(415)493-2223

– PLAN 3000 network	10,000.00

Norpak
Distributed by Apple

– Telidon Graphics	600.00

Northwest Instrument Systems Inc.
P.O. Box 1309, Beaverton, OR 97075
(800) 547-4445 (503) 297-1434

– Model 85 Ascope	995.00
– Model 65 Signal generator	850.00

Novation Inc.
18664 Oxnard St. Tarzana, CA 91356
(800) 423-5419 (818) 996-5060

– Cat (Acoustic Modem)	190.00
– D-Cat (Direct connect)	200.00

– J-Cat Originate/autoanswer miniature	150.00
– 212 Auto-Cat autodial	700.00
– 103 Smart-Cat	250.00
– 103/212 Smart-Cat	600.00
– Apple-Cat II	390.00
– Apple-Cat II 212 upgrade card	390.00
– 212 Apple-CAT II	725.00
– Expansion Module	40.00
– BSR module	20.00
– Handset	30.00
– Touch tone receiver	100.00
– Deaf Firmware	30.00

Number Nine Computer Engineering
691 Concord Ave., Cambridge, MA 02138
(617) 492-0999

– NNGS, GRFX-A2 (NEC 7220 Graphics Board)	945.00

Okidata Corp.
111 Gaither Dr., Mt. Laurel, NJ 08054
(800) OKI-DATA (609) 235-2600

– Microline 82, 120 cps, 80 col.	500.00
– Microline 84, 200 cps, 136 col.	1,400.00
– Microline 92, 160 cps, 80 col.	600.00
– Microline 93, 160 cps, 136 col.	1,000.00
– Pacemark 2410, 350 cps	3,000.00

Omega Microware
222 South Riverside Plaza, Chic. IL 60602
(800) 835-2421 (312) 648-4844

– Ramex 16K	140.00
– Ramex 128K	500.00

Orange Micro
1400 N. Lakeview Ave., Anaheim, CA 92807
(714) 779-2772

– The Grappler +	175.00
– Bufferboard	150.00
– Buffered Grappler (64K)	240.00
– Orange Printer Interface	90.00

PC Ware, Inc.
Dept. A12, 4883 Tonino Dr., San Jose, CA 95136
(408) 978-8626

– Extended //e 80 col (64K)	160.00
– Graphics Parallel	150.00
– Serial Interface	130.00
– Enhanced Serial	185.00
– General Parallel	90.00
– Centronics Parallel	80.00

Passport Designs Inc.
116 N. Catrillo Hwy., Half Moon Bay, CA 94019
(415) 726-0280

– Soundchaser Digital Keyboard	795.00
– SC-100 (Keyboard with Mountain Synthesizer)	1,190.00
– MX-5 five octave keyboard w/Turbo-Traks synthesizer	1,495.00

Percom Data Corp.
11220 Pagemill Road, Dallas, TX 75243
(800) 527-1222 (214) 747-4002

– Hard Disk	
– 5 Megabytes	1,900.00
– 10 Megabytes	2,300.00
– 15 Megabytes	2,800.00
– 20 Megabytes	3,300.00

Personal Computer Products, Inc.
16776 Bernardo Center Drive, San Diego, CA 92128
(619) 485-8411

– Appli-Card Z-80A, 4 MHz	295.00
– Appli-Card Z-80B, 6 MHz	375.00
– 64K Ram Extender	195.00
– 128K Ram Extender	295.00
– Co-processor 8088	595.00

Pion, Inc.
101R Walnut St., Watertown, MA 02172
(617) 923-8009

– Interstellar Drive (battery back 256K - 1 Megabyte)	
– 256K	1,095.00
– 256K expansions	595.00

Practical Peripherals
31245 La Baya Drive, Westlake Village, CA 91362
(818) 991-8200

– ProClock (ProDOS compatible, 1 ms IRQ)	
– SeriALL printer/modem serial card	160.00
– PRinterface	N/A
– GraphiCard	N/A
– Microbuffer II 16K	200.00
– Microbuffer II 32K	220.00
– Microbuffer II+ (Serial/Parallel/Graphics)	
– 16K	260.00
– 32K	300.00
– 64K	350.00
– Microbuffer in-line serial or parallel 32K - 256K	
– 32K	300.00
– 64K	350.00
– 64K add on modules for Microbuffer in-line	180.00
– Snapshot Option	120.00

Princeton Graphics
1101-I State Road, Princeton, NJ 08540
(800) 221-1490 (609) 683-1660

– Max 12 amber monitor	250.00
– HX-12, digital RGB 690 by 240 (480 interlaced)	700.00
– SR-12, digital RGB 690 by 480 (non-interlaced)	800.00
– RGB-80, //e 80 col./RGB board	N/A

Prometheus
45277 Fremont Blvd., Fremont, CA 94538
(415) 490-2370

– Pro Modem 1200	500.00
– Parallel Interface Card	100.00
– Graphitti Graphics Parallel	130.00
– P/S 16K buffer	199.00
– Versacard (Serial, Parallel, Clock, BSR Control)	200.00
– MEM-1 16K	100.00
– Expand-a-RAM	N/A
– 64K	375.00
– 128K	495.00
– Applesurance, disk controller and diagnostics	125.00

Quadram
4355 International Blvd., Norcross, GA 30093
(404) 923-6666

– eRAM 80 (//e 64K and 80 column)	160.00
– Quadchrome, 690 dots by 480 lines, digital RGB	790.00
– APIC (Parallel Interface Card)	140.00
– APIC/G (Parallel Interface Card)	160.00
– Microfazer buffer	N/A
– 64K	300.00
– 256K	900.00
– 512K	1,400.00
– Quadlink w/controller for IBM PC drives	680.00
– Quadjet (color ink jet printer, 640 dots/line)	895.00

Quality Software
21601 Marilla St., Chatsworth, CA 91311
(818) 709-1721

– *Understanding the Apple II*	23.00
– *Beneath Apple DOS*	N/A

Quentin Research, Inc.
9207 Eton Ave., Chatsworth, CA 91311
(213) 709-6500

– AP-100 Disk II compatible drive	335.00
– AP-105 half height Disk II compatible	295.00
– AP-110 twin half height Disk II compatible w/controller	600.00
– Q-500 Hard disk	N/A

– 5 Megabytes	N/A
– 10 Megabytes	1,995.00
– 15 Megabytes	2,295.00

Qume
2350 Qume Drive, San Jose, CA 95131
(408) 942-4000

– Sprint 11 Plus, 40 cps	1,685.00
– Sprint 11/55 Plus, 55 cps.	1,990.00
– Sprint 11/40-130, 130 character print wheel	2,965.00
– Letter Pro 20 cps	N/A

Qwerty, Inc.
9252 Chesapeake Dr., Suite 600, San Diego, CA 92123
(619) 569-5283

– Q-Pak 68 development system 68008 uses Apple motherboard memory	695.00

RB Robot Corp.
18301 West 10th Ave., Suite 310, Golden, CO 80401
(303) 279-5525

– RB5X Robot	1,795.00

RC Electronics
5386 Hollister Ave., Santa Barbara, CA 93111
(805) 964-6708

– Bus Rider Logic Analyzer	395.00
– Applescope APL-D2	795.00
– Applescope HR-14	995.00
– Applescope HS-7	N/A

RH Electronics
566 Irelan Street, Buellton, CA 93427
(805) 688-2047

– Super Fan II	75.00
– Super Fan w/Zener Ray 2	110.00
– Guardian Angel	595.00

Rana
21300 Superior St., Chatsworth, CA 91311
(800) 421-2207 (818) 709-5484

– 8086/2 (IBM PC and Disk II compatible)	1,795.00
– Elite One (163K Drive)	380.00
– Elite Two (320K Drive)	650.00
– Elite Three (640K Drive)	850.00
– Four drive Controller Card	100.00
– 2.5 Megabyte Floppy	1,500.00

Rhino Robots
2505 South Neil St., Champaign, IL 61820
(217)352-8485

– XR-2 robot	N/A

Roland Corp.
2401 Saybrook Ave., Los Angeles, CA 90040
(213) 685-5141

– Compumusic CMU-800	500.00

SSM Microcomputer Products Inc. (Transend)
2190 Paragon Dr., San Jose, CA 95131
(408) 946-7400

– Prototyping Board	25.00
– Extender Board	30.00
– IEEE-488 Controller	475.00
– ASIO	150.00
– AIO-II	225.00
– APPIC/G Graph-It parallel	110.00
– Modemcard	325.00
– Transmodem 1200	700.00

STB Systems Inc.
601 North Glenville, Suite 125, Richardson, TX 75081
(214) 234-8750

– STB 16K	170.00
– STB 16/64K	250.00
– STB 64K	N/A
– STB 128K	400.00
– STB-80 Video	250.00

Saturn Systems Inc. (Titan Technologies)
P.O. Box 8050, Ann Arbor, MI 48107
(313) 973-8422

– 32K Ram Board	220.00
– 64K Ram Board	350.00
– 128K Ram Board	500.00
– Neptune Board (//e)	
64K	250.00
– 128K	400.00
– 192K	500.00
– Accelerator	600.00

Scooter
746 Vermont St., Palatine, IL 60067
(312) 359-6040

– 0-Force X-Port	20.00
– 0-Force Twin Port	35.00

Scott Instruments
815 North Elm, Denton, TX 76201
(817) 387-9514

– Shadow VET	595.00
– VET2	795.00

Segull (Nibble 4/8 p. 137)
P.O. Box 2724, Taunton, MA 02780
(617) 823-9684

– Pro-Tech locking Apple stand	145.00

Siemens Corp.
240 E. Palais Rd., Anaheim, CA 92805
(714) 991-9700

– PT-88 Ink Jet Printer, 150 cps	895.00
– 2712 Ink Jet Printer, 270 cps	2,250.00

Snave Systems
P.O. Box 957, Niles, IL 60648
(312) 966-4505

– Fly Board	130.00

Sorrento Valley Associates
11722 Sorrento Valley Road, San Diego, CA 92121
(714) 452-0101

– App-L-Cache (256K DE) (out of production)	995.00
– ZVX4 8 inch controller (Out of production)	445.00
– Megaflex (out of production)	300.00

SCRG (Southern California Research Group
P.O. Box 2231-S, Goleta, CA 93118
(805) 685-1931

– Switch-a-slot	180.00
– Extend-a-slot	35.00

Softalk
P.O. Box 7039, North Hollywood, CA 91605
(800) 821-6231 (818) 980-5074

– magazine; one year subscription	24.00

Southwestern Data Systems (Roger Wagner Publishing)
10761 Woodside Ave., Suite E, P.O.Box 582, Santee, CA 92071
(800) 421-6526 (619) 562-3221

– 65C02 Upgrade	35.00
– Routine Machine ampersand package	65.00
– Merlin assembler	65.00
– Speedstar compiler	135.00
– Apple-DOC	40.00
– Listmaster w/Applespeed	40.00

Speis Laboratories
P.O. Box 336, Lawndale CA 90260
(213) 644-0056

– Super MX Wordstar/dot matrix interface	175.00

Star Micronics
2803 N.W. 12th St., Dallas/Ft. Worth Airport, TX 75261

– Gemini 10X, 120 cps, 80 column	400.00
– Gemini 15X, 120 cps, 132 column	650.00
– Delta 10, 160 cps	650.00
– STX-80, 60 cps	200.00
– Powertype letter quality, 18 cps	N/A
– Radix-15, 200cps	N/A

Stellation Two
Box 2343, Santa Barbara, CA 93120
(805) 966-1140

– The Mill 6809	295.00
– VitaMill w/64K RAM	450.00
– 68008 co-processor	230.00

Street Electronics
1140 Mark Ave., Carpinteria, CA 93013
(805) 684-4593

– Echo II Speech Synthesizer	150.00

Strobe Inc.
897 Independence Ave., Bldg. 5A, Mountain View, CA 94043
(415) 969-5130

– M100 single pen drum plotter	595.00
– M260 6 pen plotter	995.00

Summagraphics
35 Brentwood Ave., Fairfield, CT 06430
(203) 384-1344

– Graphics Tablet	N/A

Suncom
650E Anthony Trail, Northbrook, IL 60062
(312) 291-9780

– Starfighter joystick	50.00

Sweet Micro Systems
150 Chestnut Street, Providence, RI 02903
(401) 273-5333

– Mockingboard	
– Speech I	100.00
– Sound II	130.00
– Sound/Speech I	180.00

Symtec
15933 West 8 Mile, Detroit, MI 48235
(313) 272-2950

– PGS III graphics generator	
– 16 color 256 by 240	4,500.00
– 4,000 color 512 by 480	7,000.00
– 4,000 color 512 by 2 pages	8,500.00
– 4,000 color 512 by 2 pages w/preview	10,000.00
– Frame Grabber 512 by 480, 16 gray levels	3,000.00

Synetix
15050 N.E. 95th St., Redmond, WA 98052
(800) 426-7412 (206) 881-7110

– Sprite I	150.00
– Sprite II (with sound)	250.00
– Supersprite (sound and voice)	400.00
– SSD Solid State Disk Emulator 144K (Flashcard)	350.00
– SSD Solid State Disk Emulator 288K	530.00

Syntauri Corporation
4962 El Camino Real, Suite 112, Los Altos, CA 94022
(800) 227-1817

– AS-05C (5 octave keyboard, Mountain Synth, Ham. B-3)	1,495.00
– SM-05 (5 octave keyboard, Mountain Synth	1,295.00
– SM-04 (4 octave keyboard, Mountain Synth	995.00

TG Products
1104 Summit Ave., Suite 110, Plano, TX 75074
(800) 531-5555 (214) 424-8568

– Game Paddles	40.00
– Joystick	65.00
– Select-a-port	60.00
– Trackball	65.00

Taurus Computer Systems, Inc.
7533 Republic Court #201, Alexandria, VA 22306
(703) 765-5829

– 8-inch controller and single drive	888.00
– 8-inch controller and dual drive	1,488.00

Taxan
TSK Electronics: 18005 Cortney Court, City of Industry, CA 91748
(213) 810-1291

– RGB2 interface	140.00
– Model 410-80 (//e 80 column RGB)	200.00
– Model 410-64 (//e 80 column w/64K, RGB)	350.00
– RGBvision I, 380 lines	400.00
– RGBvision III, 630 lines, 18 MHz	630.00
– Model 210, composite color, 380 lines	400.00

Tech Design
3638 Grosvenor Drive, Ellicott City, MD 21043

– Adam & Eve Game Paddles	35.00
– Adam & Eve Joystick	50.00

Telemax
780 Lorraine Drive, Box 339, Warrington, PA 18976
(215) 343-3000

– VCB 24e	190.00
– VCB 24+	190.00

Tex Print Inc.
8 Blanchard Road, Burlington, MA 01803
(800) 255-1510 (617) 273-3384

– Print It	300.00
– Model 2	175.00

Thirdware Computer Products
4747 NW 72nd Avenue, Miami, FL 33166
(800)528-6050

– FingerPrint (screen grab NMI printer card)	150.00

Third Millenium Engineering
1015 Gayley Ave., Suite 394, Los Angeles, CA 94816
(213) 473-2102

– The Arcade Board	300.00

Thunderware Inc.
44 Hermosa Ave., Oakland, CA 94618
(415) 652-1737

– Thunderclock	150.00
– X-10 interface	50.00

Timecor
Four Longfellow Place, P.O. Box 8928, Boston, MA 02114
(800) 824-7888 (617) 720-4090

– The Operator (modem card)	160.00

Titan Technologies (see Saturn Systems)

Toshiba America, Inc.
2441 Michelle Dr., Tustin, CA 92680
(800) 457-7777 (714) 730-5000

– P1350, 190 cps dot/letter quality	2,195.00
– P1340, 100 cps dot/letter quality	995.00

Track House
625 Trailwood Ct., Garland, TX 75043
(214) 270-0922

– Programmable //e Tender	200.00
– Standard //e Tender	150.00
– Standard II/II+ Tender	150.00

Transend (see SSM Microcomputer Products)

Transtar (Seikosha)
P.O. Box C-96975, Bellevue, WA 98009
(206) 454-2950

– Transtar 120 (portable daisy wheel) (14 cps Diablo 1610 code compatible)	600.00
– Transtar 130 18 cps, 1610 compatible	900.00
– Transtar 140 40 cps, 1610 code compatible	1,700.00
– Transtar 315 color dot matrix	600.00

U.S. Robotics
1123 West Washington Blvd., Chicago, IL 60607
(312) 733-0497

– Password 1200 baud modem	450.00

USI
71 Park Lane, Brisbane, CA 94003
(415) 468-4900

– Autodial/autoanswer modem external 300 baud	100.00
– 900/G (Pi1) 9 inch green monitor	160.00
– 1200/G (Pi2) 12 inch green monitor	200.00
– 1200/A (Pi3) 12 inch amber monitor	240.00
– 900/A (Pi4) 9 inch amber monitor	200.00
– Color 1400C, 260 Horiz, 40 by 25	400.00
– MJ-22 Technica 500 by 240 color	500.00
– AppleMod RF modulator	35.00

U-Microcomputers
Winstanley Industrial Estate, Long Lane, Warrington,
Cheshire, WA2 8PR, England, U.K.

300 Broad St., Stamford, CT 06901
(203) 359-4236 (800) 243-2475

– A/D converter	525.00
– Digital I/O and timer	160.00
– Eight serial port card	N/A
– U-Print	200.00
– U-Centronics	80.00
– U-Net network controller	N/A
– U-Z80	140.00
– U-Term, 80 col.	225.00
– U-RAM 128K	450.00

VR Data
777 Henderson Boulevard N-6, Folcroft, PA 19032
(800) 345-8102 (215) 461-5300

– Removable Hard Disk 5 Megabytes	1,495.00
– Dual 10 Megabytes hard disk	2,000.00

Vector
12460 Gladstone Ave., Sylmar, CA 91342
(213) 365-9661

– 4609 Plugboard prototyping board	25.00

Verbatim

– Disk Drive Analyzer	40.00

Versa Computing, Inc.
3541 Old Conejo Rd., Suite 104, Newbury Park, CA 91320
(805) 498-1956

– VersaWriter graphics tablet	300.00
– EZ Port Game I/O Extender	25.00
– EZ Port II Zero Insertion Force	N/A

Video Associates Labs
2304 Hancock Dr., Suite One, Austin, TX 78757
(512) 459-5684

– VB3 NTSC video, w/genlock and keyover	2,400.00
– VB3e	2,600.00

Videx
1105 N.E. Circle Blvd., Corvalis, OR 97330
(503) 758-0521

– Videoterm Card	280.00
– Videoterm w/softswitch	310.00
– Videoterm w/softswitch and inverse	320.00
– Videoterm Switchplate	20.00
– Character Set EPROMs	30.00
– Ultraterm	380.00
– Enhancer II	150.00
– Keyboard and Display Enhancer	130.00
– Function Strip	80.00
– PSIO	230.00
– Uniprint	90.00

Vista
1317 E. Edinger, Santa Ana, CA 92705
(800) 854-8017 (714) 953-0523

– Vision-20 character generator	N/A
– Vision-40 special character generator	N/A
– Vision-80 column	N/A

– Model 150 Type Ahead Buffer	50.00
– A800 Controller	379.00
– A800 + dual SSDD, V1000 cabinet (1.2 Megabyte total)	1,500.00
– A800 + dual DSDD, V1000 cabinet (2.4 Megabytes total)	1,800.00
– A800 + dual DSDD slimline, V1100 cabinet	2,100.00
– A801 + V1200 6MB floppy	1,500.00
– Solo (Disk II compatible)	300.00
– Duet thinline (320K)	425.00
– Quartet (2 by 320K)	850.00

Voice Machine Communications (The Voice Connection)
17835 Skypark Circle, Irvine, CA 92714
(714) 261-2366

– Voice Input Module 1	845.00
– VIM 2 (w/gooseneck microphone)	920.00
– VIM 1e	920.00
– VIM 2e	995.00
– Wireless microphone	595.00
– Headset	180.00

Voicetek
Dept. G, P.O. Box 388, Goleta, CA 93116
(805) 685-1854

– Cognivox	250.00

Votrax
500 Stephenson Highway, Troy, MI 48084
(800) 521-1350 (313) 588-0341

– Type n'Talk Speech	250.00
– Personal Speech System	400.00

Vufax, Inc.
5301 Covington Highway, Decatur, GA 30035
(800) 241-1119 (404) 981-6778

– Infax 101A 10 Meg hard disk cartridge	2,395.00

Vynet
160B Albright Way, Los Gatos, CA 95030
(408) 370-0555

– V101A Interactive Telephone Interface	295.00
– V200-VSM LPC Synthesis module	150.00
– V200-SPM ADPCM Speech playback	250.00
– V300-LSM Low speed modem module	150.00
– V400-SAM ADPCM record module	1,000.00

WICO
6400 Gross Point Rd., Niles, IL 60648

– Trackball	110.00
– Joystick	50.00

Wesper Microsystems
14321 New Myford Rd., Tustin, CA 92680
(714) 730-6250

– Wizard-80	250.00
– Wizard-16K	95.00
– BPO 16K	180.00
– BPO 32K	220.00
– SOB 16K	250.00
– SOB 32K	290.00

Western Design Center
2166 East Brown Rd., Mesa, AZ 85203
(602) 962-4545

– 65816 tool box	1,495.00
– 65816 based CAD chip design system	5,000.00

X-Comp, Inc.
7566 Trade Street, San Diego, CA 92121
(619) 271-8730

– The Toaster (2 by 5 Megabyte removable)	2,795.00

Xebec
432 Lakeside Dr., Sunnyvale, CA 94086
(408) 733-4200 (800) 538-1644 (800) 672-1842

– Hard Disk kit	1,300.00

Zenith (Heath Co.)
Benton Harbor, MI 49022

– Hero I Robot	2,500.00

Zicor
2296 Cascade Plaza North, Woodbury, MN 55125
(612) 731-1762

– Omega II (keyboard)	400.00

Zoom Telephonics
207 South Street, Boston, MA 02111
(617) 423-1288 (800) 631-3116

– Networker 300 baud direct connect	130.00

Appendix G

Listing by Subject

This appendix has been provided as a general information guide. The products listed and the prices are subject to change. Due to the constant fluctuations in the industry, neither the prices nor the products listed here can be guaranteed for accuracy. Where no price was available at the time of publishing, an N/A symbol has been substituted.

A. Video Display Hardware

1. RF Modulators
2. Character ROMs
3. Monochrome Monitors
4. //e 80 Column Cards
5. 80 Column and Terminal Cards
6. RGB Boards
7. Graphics Generators
8. NTSC Output and Effects Generators
9. Color Monitors
10. Video Digitizers
11. Light Pens

1. RF Modulators

Advanced Logic Systems	
– Dirt Cheap Video, 64 column w/RF modulator for TV	90.00
Apple Computer	
– RF Modulator	N/A
M & R Enterprises	
– Sup'R Mod RF Modulator	70.00
USI	
– Applemod RF modulator	35.00

2. Character ROMs

Dan Paymar	
– Lowercase Adapter	37.50
– Lowercase Adapter-Rev. 7	27.50

Lazerware	
– Lowercase Plus	60.00
– Keyboard Plus	30.00
MPC Peripherals	
– AP-96 Lower case generator	29.50
Vista	
– Vision-20 character generator	N/A
– Vision-40 special character generator	N/A

3. Monochrome Monitors

Amdek	
– Amdek Video 300A amber monitor	200.00
– Amdek Video 300 green monitor	180.00
Apple Computer	
– Monitor II	230.00
– Monitor III	260.00
BMC	
– BM-12 AU monochrome monitor, 15 MHz	130.00
Comrex International, Inc.	
– CR-5400 20MHz, 800 line (P31, P39, or PDB; amber)	100.00
– CR-5600 20MHz, 1,000 line (P31, P39)	135.00
– CR-5600 20MHz, 1,000 line amber	150.00
Micro Display Systems	
– The Genius 57 line display	1,795.00
Mitac	
– Monitor	130.00
NEC Home Electronics	
– 1201 Hi Res Green Monitor	285.00
– 1205 Hi Res Amber Monitor	210.00
Princeton Graphics	
– Max 12 amber monitor	250.00
USI	
– 900/G (Pi1) 9 inch green monitor	160.00
– 1200/G (Pi2) 12 inch green monitor	200.00
– 1200/A (Pi3) 12 inch amber monitor	240.00
– 900/A (Pi4) 9 inch amber monitor	200.00

4. //e 80 column cards

Amdek	
– DVM 80e RGB board	195.00
Apple Computer	
– //e text card	125.00
– //e 64K extended text card	295.00

Applied Engineering	
– Memory Master //e (64K)	170.00
– Memory Master //e (128K)	250.00
Coex (Components Express)	
– //e Extended RAM card	200.00
ComX	
– //e 64K, 80 Col.	300.00
MPC Peripherals	
– Ramtext //e (64K)	290.00
Micro Sci	
– 80/64e //e 80 column card	170.00
Micromax	
– Viewmax-80e (64K)	190.00
– Viewmax-80e (128K)	300.00
Microsoft Corp.	
– Premium SoftCard //e Z-80B	495.00
Microtek	
– Magnum 80e	50.00
– Magnum 80Me (64K)	140.00
PC Ware, Inc.	
– Extended //e 80 column (64K)	160.00
Princeton Graphics	
– RGB-80, //e 80 column RGB board	N/A
Quadram	
– eRAM 80 (//e 64K and 80 column)	160.00
Saturn Systems Inc. (Titan Technologies)	
– Neptune Board (//e)	
– 64K	250.00
– 128K	400.00
– 192K	500.00
Taxan	
– Model 410-80 (//e 80 column RGB)	200.00
– Model 410-64 (//e 80 column w/64K, RGB)	350.00

5. 80 Column and Terminal Cards

Advanced Logic Systems	
– Smarterm II	180.00
– Dirt Cheap Video, 64 column w/RF modulator for TV	90.00
– Smarterm I	345.00
– Version 2.0 ROM	40.00
Applied Engineering	
– Viewmaster 80 (seven by nine)	170.00

M & R Enterprises	
– Sup'R Terminal	350.00
Micro Display Systems	
– The Genius 57 line display	1,795.00
Micromax	
– Viewmax-80	230.00
– Ultimax	N/A
Microtek	
– Magnum 80	200.00
STB Systems Inc.	
– STB-80 Video	250.00
U-Microcomputers	
– U-Term, 80 column	225.00
Videx	
– Videoterm Card	280.00
– Videoterm w/softswitch	310.00
– Videoterm w/softswitch and inverse	320.00
– Videoterm Switchplate	20.00
– Character Set EPROMs	30.00
– Ultraterm	380.00
Vista	
– Vision-80 column	N/A
Wesper Microsystems	
– Wizard-80	250.00

6. RGB Boards

Advanced Logic Systems	
– Color II (RGB Interface)	180.00
Amdek	
– DVM II RGB board	180.00
– DVM 80e RGB board	195.00
– DVM III multiple mode RGB board	195.00
Electrohome	
– Supercolor analog RGB (out of production)	N/A
Microtek	
– Rainbow 256 (analog RGB board)	150.00
– Rainbow 16 A	120.00
Princeton Graphics	
– RGB-80, //e 80 column RGB board	N/A
Taxan	
– RGB2 interface	140.00
– Model 410-80 (//e 80 column RGB)	200.00
– Model 410-64 (//e 80 column w/64K, RGB)	350.00

Telemax	
– VCB 24e	190.00
– VCB 24+	190.00

7. Graphics Generators

ALF	
– AD8088 Processor card with FTL and MET	345.00
– Expansion Board for AD8088 (sockets for 64/128K)	295.00
– ADGS	N/A
Micrographic Images	
– NEC 7220 board	N/A
Norpak	
– Telidon Graphics	600.00
Number Nine Computer Engineering	
– NNGS, GRFX-A2 (NEC 7220 Graphics Board)	945.00
Rana	
– 8086/2 (IBM PC and Disk II compatible)	1,795.00
Symtec	
– PGS III graphics generator	
16 color 256 by 240	4,500.00
4,000 color 512 by 480	7,000.00
4,000 color 512 by 480; 2 pages	8,500.00
4,000 color 512 by 480; 2 pages w/preview	10,000.00
Synetix	
– Sprite I	150.00
– Sprite II (with sound)	250.00
– Supersprite (sound and voice)	400.00
Third Millenium Engineering	
– The Arcade Board	300.00
Western Design Center	
– 65816 based CAD chip design system	5,000.00

8. NTSC Output and Effects Generators

Adwar	
– ARS-170A (Apple-NTSC graphics converter)	1,495.00
– ARS-170AX with genlock	1,995.00
– Apple Proc Mod	100.00
Symtec	
– PGS III graphics generator	
16 color 256 by 240	4,500.00
4,000 color 512 by 480	7,000.00
4,000 color 512 by 480; 2 pages	8,500.00
4,000 color 512 by 480; 2 pages w/preview	10,000.00

Video Associates Labs	
– VB3 NTSC video, w/genlock and keyover	2,400.00
– VB3e	2,600.00

9. Color Monitors

Amdek	
– Color I 260 by 300 lo-res, composite	380.00
– Color I+ 260 by 300 lo-res (non-glare screen)	400.00
– Color II+ 560 by 240, digital RGB, med res	560.00
– Color III 260 by 300 lo-res, digital RGB	450.00
– Color III+	480.00
– Color IV 720 by 420, 16 MHz, analog color range	690.00
Comrex International, Inc.	
– CR-6500 RGB 300 by 260	325.00
NEC Home Electronics	
– 1212 Composite, lo-res color	400.00
– 1216 RGB hi-res color, 640 by 240, 10 MHz	600.00
Princeton Graphics	
– HX-12, digital RGB 690 by 240, (480 interlace), 18 MHz	700.00
– SR-12, digital RGB 690 by 480, (non-interlace), 25 MHz	800.00
– RGB-80, //e 80 col./RGB board	N/A
Quadram	
– Quadchrome, 690 dots by 480 lines, digital RGB	790.00
Taxan	
– RGBvision I, 380 lines, 15 MHz	400.00
– RGBvision III, 630 lines, 18 MHz	630.00
– Model 210, composite color, 380 lines	400.00
USI	
– Color 1400C, 260 Horiz	400.00
– MJ-22 Technica 500 by 240	500.00

10. Video Digitizers

Computer Stations	
– Dithertizer (camera, dithertizer, printer interface)	1,175.00
Micro Works	
– DS-65 Digisector digitizing TV camera	600.00
– digisector board only	350.00
Micrographic Images	
– high speed flash video digitizer	N/A
Symtec	
– Frame Grabber 512 by 480, 16 gray levels	3,000.00

11. Light Pens

Gibson Laboratories, Inc.	
– Light Pen LPS II	350.00
Magellan	
– Light Pen	190.00

B. Keyboards, pointers and digitizers

1. Enhancements for Built-In Keyboard
2. Numeric and Function Keypads
3. Detachable Keyboards
4. Bar Code and Card Readers
5. Lowercase ROMs
6. Pointing Devices
7. Game Controls
8. Game Port Expansion
9. Digitizing Tablets
10. Video Digitizers

1. Enhancements for Built-In Keyboard

High Order Microelectronics	
– Repeaterrr	27.00
– Repeaterrr +	37.00
Innovative Micro Goodies	
– D-Tach keyboard carrier	90.00
Keytec	
– keyboard enclosure	60.00
Vista	
– Model 150 Type Ahead Buffer	50.00

2. Numeric and Function Keypads

Advanced Business Technology	
– Keypad 13 numeric keys	125.00
– Keypad 13 numeric keys for //e	145.00
– Softkey 15 function keys	150.00
Apple Computer	
– Numeric key pad	160.00
Creative Computer Peripherals, Inc.	
– Keywiz 83	300.00
– Keywiz Convertible	300.00
– Keywiz VIP	440.00

Track House	
– Programmable //e Tender	200.00
– Standard //e Tender	150.00
– Standard II/II+ Tender	150.00
Videx	
– Enhancer II	150.00
– Keyboard and Display Enhancer	130.00
– Function Strip	80.00

3. Detachable Keyboards

Amkey	
– PRO 100 keyboard	295.00
EPS (Executive Peripherals)	
– Detachable keyboard	400.00
Key Tronic	
– KB-200	300.00
Multitech Electronics	
– Accu-Feel MAK-II keyboard	150.00
Zicor	
– Omega II (keyboard)	400.00

4. Bar Code and Card Readers

Advanced Business Technology	
– Symbtrak Bar Code Reader and Printer	500.00
– Retail Manager	2,200.00
– Bar Code wand	225.00
Chatsworth Data Corporation	
– 2000 series card reader	1,675.00
– 500 series hand fed card reader	1,095.00
Mountain Computer	
– Model 1100 A Card Reader	1,500.00

5. Lowercase Character ROMs

Dan Paymar	
– Lowercase Adapter	37.50
– Lowercase Adapter—Revision 7	27.50
Lazerware	
– Lowercase Plus	60.00
– Keyboard Plus	30.00
MPC Peripherals	
– AP-96 Lower case generator	29.50

Vista	
– Vision-20 character generator	N/A
– Vision-40 special character generator	N/A

6. Pointing Devices

Apple Computer	
– AppleMouse	150.00
Chalkboard	
– Power Pad	100.00
CTA (Computer Technology Associates)	
– Touch Screen Bezel	695.00
Gibson Laboratories, Inc.	
– Light Pen LPS II	350.00
Koala Technologies Corp.	
– Touch Pad	125.00
Magellan	
– Light Pen	190.00

7. Game Controls

Apple Computer	
– Joystick	60.00
– Paddle	29.00
Hayes Products	
– Mach II Joystick	40.00
– Mach III Joystick w/fire button	50.00
Kraft	
– Premium Joystick	65.00
– Premium Game Paddle Pair	50.00
Mimco	
– Joystick	60.00
Suncom	
– Starfighter joystick	50.00
TG Products	
– Game Paddles	40.00
– Joystick	65.00
– Select-a-port	60.00
– Trackball	65.00
Tech Design	
– Adam & Eve Game Paddles	35.00
– Adam & Eve Joystick	50.00
WICO	
– Trackball	110.00
– Joystick	50.00

8. Game Port Expansion

Scooter	
– 0-Force X-Port	20.00
– 0-Force Twin Port	35.00
TG Products	
– Select-a-port	60.00
Versa Computing, Inc.	
– EZ Port Game I/O Extender	25.00
– EZ Port II Zero Insertion Force	N/A

9. Graphics Tablets

Apple Computer	
– Graphics Tablet	800.00
Bausch and Lomb	
– HiPad DT-114	
Micro Control Systems	
– Space Tablet 3-axis digitizer	475.00
Summagraphics	
– Graphics Tablet	
Versa Computing, Inc.	
– VersaWriter graphics tablet	300.00

10. Video Digitizers

Computer Stations	
– Dithertizer (camera, dithertizer, printer interface)	1,175.00
Micro Works	
– DS-65 Digisector digitizing TV camera	600.00
– digisector board only	350.00
Micrographic Images	
– Video Digitizer	N/A
Symtec	
– Frame Grabber, 512 by 480, 16 gray levels	3,000.00

C. Voice and Music

1. Music
2. Sound Effects
3. Speech Synthesis
4. Speech Recognition

1. Music

ALF	
– MC1 Music Card nine voice music synthesizer	195.00
– MC16 Music Card three voice music synthesizer	195.00
Applied Engineering	
– Super Music Synthesizer 16 voice	160.00
Fender/Rogers/Rhodes	
– Rhodes Chroma Music Terminal, 16 voices	5,295.00
– additional 16 voices	3,150.00
– Apple interface kit	495.00
Mountain Computer	
– Music System	395.00
Passport Designs Inc.	
– Soundchaser Digital Keyboard	795.00
– SC-100 (Keyboard with Mountain Synthesizer)	1,190.00
– MX-5 five octave keyboard w/Turbo-Traks synthesizer	1,495.00
Roland Corp.	
– Compumusic CMU-800	495.00
Syntauri Corporation	
– AS-05C (5 octave keyboard, Mountain Synthsizer, Ham. B-3)	1,495.00
– SM-05 (5 octave keyboard, Mountain Synthesizer, Ham. B-3)	1,295.00
– SM-04 (4 octave keyboard, Mountain Synthesizer, Ham. B-3)	995.00

2. Sound Effects

Decillionix	
– DX-1 Sound Processing System	240.00
E-mu Systems	
– Drumulator	995.00
Sweet Micro Systems	
– Mockingboard	
Sound II	130.00
Sound/Speech I	180.00
Synetix	
– Sprite II (with sound)	250.00
– Supersprite (sound and voice)	400.00
Third Millenium Engineering	
– The Arcade Board	300.00

3. Speech Synthesis

Alien Group	
– Voice Box	215.00

Artra, Inc.	
– Waldo Voice Activated Control System	600.00
w/"robot voice" unlimited vocabulary	+199.00
w/"human quality voice" 206 words	+199.00
BSR direct connect	70.00
Don't Ask	
– S.A.M. Software Automatic Mouth (includes D/A converter and amplifier)	150.00
Micro Mint	
– Sweet Talker	100.00
– Micromouth	150.00
– Microvox	300.00
Mountain Computer	
– Supertalker SD 200	200.00
Multitech Electronics	
– SSB-Apple speech synthesizer	200.00
RB Robot Corp.	
– RB5X Robot	1,795.00
Street Electronics	
– Echo II Speech Synthesizer	150.00
Sweet Micro Systems	
– Mockingboard	
Speech I	100.00
Sound II	130.00
Sound/Speech I	180.00
Voicetek	
– Cognivox	250.00
Votrax	
– Type n' Talk Speech	250.00
Vynet	
– V101A Interactive Telephone Interface	295.00
– V200-VSM LPC Sythesis module	150.00
– V200-SPM ADPCM Speech playback	250.00
– V300-LSM Low speed modem module	150.00
– V400-SAM ADPCM record module	1,000.00

4. Speech Recognition

RB Robot Corp.	
– RB5X Robot	1,795.00
Scott Instruments	
– Shadow VET	595.00
– VET2	795.00

Voice Machine Communications	
– Voice Input Module 1	845.00
– VIM 2 (w/gooseneck microphone)	920.00
– VIM 1e	920.00
– VIM 2e	995.00
– wireless microphone	595.00
– headset	180.00
Voicetek	
– Cognivox	250.00

D. System Support

1. Power Protection
2. High Amperage Power Supply
3. Fans
4. Cases and Stands
5. Electronic Prototyping and Testing Aids
6. Bus and Port Expansion

1. Power Protection

DOSS Industries	
– Apple Center, case w/fan and surge protector	240.00
Electronic Data Protection	
– Lemon (EC-I)	60.00
– Lime (EC-II)	90.00
– Peach (EC-IV)	100.00
– Orange (EC-V)	140.00
– Kiwi	
– Hawk power monitor	200.00
– Groundhog anti-static mat	90.00
– Grizzly 200 watt	900.00
– Grizzly 500 watt	2,400.00
– Grizzly 1,000 watt	5,200.00
Kemcore Company	
– Fan and back panel	155.00
Kensington Microware	
– System Saver Fan	90.00
RH Electronics	
– Super Fan w/Zener Ray 2	110.00
– Guardian Angel	595.00

2. High Amperage Power Supply

M & R Enterprises	
– Sup'R Switcher	295.00

3. Fans

DOSS Industries	
– Apple Center, case w/fan and surge protector	240.00
FMJ	
– Sentry II Cool Stack	175.00
Kemcore Company	
– Fan and back panel	155.00
Kensington Microware	
– System Saver Fan	90.00
M & R Enterprises	
– Sup'R Fan	50.00
RH Electronics	
– Super Fan II	75.00
– Super Fan w/Zener Ray 2	110.00

4. Cases and Stands

DOSS Industries	
– Apple Center, case w/fan and surge protector	240.00
FMJ	
– Sentry II Cool Stack	175.00
Segull	
– Pro-Tech locking Apple stand	145.00

5. Electronic Prototyping and Testing Aids

Apple Computer	
– Hobby/Prototyping Board	N/A
California Computer Systems	
– Wire Wrap Board (out of production)	N/A
– Soldertail Board (out of production)	N/A
– Extender Board (out of production)	N/A
– Etch Board (out of production)	N/A
Coex (Components Express)	
– Protocard	30.00
– Extender card	30.00
Douglas Electronics	
– Appleseed motherboard BX DE-12	95.00
– Extender Board 6 DE-12	18.00
– General Purpose Bread Board 25 DE-12	24.00
– Wire Wrap Panel 32 DE-12	21.00
Hollywood Hardware	
– PRO-1 prototype board	30.00

RC Electronics
- Bus Rider Logic Analyzer 395.00

SSM Microcomputer Products Inc. (Transend)
- Prototyping Board 25.00
- Extender Board 30.00

Vector
- 4609 Plugboard prototyping board 25.00

6. Bus and Port Expansion

Legend Industries N/A
- Slot 8

Mountain Computer
- Expansion Chassis 750.00

Scooter
- 0-Force X-Port 20.00
- 0-Force Twin Port 35.00

SCRG (Southern California Research Group
- Switch-a-slot 180.00
- Extend-a-slot 35.00

TG Products
- Select-a-port 60.00

Versa Computing, Inc.
- EZ Port Game I/O Extender 25.00
- EZ Port II Zero Insertion Force N/A

E. Clocks and Lab Equipment

1. Real Time Clocks
2. BSR Control
3. Board Level A/D Converters
4. Oscilloscope Systems
5. Board Level Lab System Components
6. Full Scale Lab Systems
7. Specialized Input and/or Control Systems
8. Robots

1. Real Time Clocks

Applied Engineering
- Timemaster 130.00

Artra, Inc.
- Waldo Voice Activated Control System 600.00

California Computer Systems	
– Model 7424 Calendar/Clock Module	115.00
– Model 7440 Programmable Timer	120.00
Hayes	
– Chronograph	N/A
MBI	
– Appletime Clock Card	85.00
Mountain Computer	
– CPS Multifunction (out of production)	240.00
– The Clock	280.00
Practical Peripherals	
– ProClock (ProDOS compatible, 1 ms IRQ)	N/A
Prometheus	
– Versacard (Serial, Parallel, Clock, BSR Control)	199.00
– Pro-Modem 1200	500.00
Thunderware Inc.	
– Thunderclock	150.00

2. BSR Control

Artra, Inc.	
– Waldo Voice Activated Control System	600.00
– BSR direct connect	70.00
Bi Comm Systems	
– PC-1 power line controller	265.00
Novation Inc.	
– Apple-Cat II modem	390.00
– Expansion Module	40.00
– BSR module	20.00
Prometheus	
– Versacard (Serial, Parallel, Clock, BSR Control)	199.00
Thunderware Inc.	
– Thunderclock	150.00
– X-10 interface	50.00

3. Board Level A/D Converters

Applied Engineering	
– A/D converter, 8 bit, 8 channel	130.00
– A/D converter, 12 bit, 16 channel	320.00
California Computer Systems	
– Modle 7470 Analog to Digital Converter	105.00

Data Translations	
– DT2832 A/D Converter	695.00
5714	+550.00
5716	+975.00
– DT2834 A/D Converter	795.00
– DT710 screw terminal panel	190.00
Hollywood Hardware	
– AD 121602	300.00
Interactive Structures	
– DAISI Data Acquisition and Instrumentation Systems Interface	
– AI13 (12 bit analog input)	550.00
– AI02 (8 bit analog input)	299.00
Mountain Computer	
– A/D + D/A	350.00
U-Microcomputers	
– A/D converter	525.00

4. Oscilloscope Systems

Northwest Instrument Systems Inc.	
– Model 85 Ascope	995.00
– Model 65 Signal generator	850.00
RC Electronics	
– Applescope APL-D2	795.00
– Applescope HR-14	995.00
– Applescope HS-7	N/A

5. Board Level Lab System Components

California Computer Systems	
– Model 7440 Programmable Timer	120.00
Hollywood Hardware	
– PD48 Buffered digital interface, 48 lines	250.00
– A16G op amp module	80.00
Interactive Microware	
– Adalab	495.00
Micro Dimensions	
– MicroD-1000 lab interface	300.00
Mountain Computer	
– A/D + D/A	350.00
Snave Systems	
– Fly Board	130.00
U-Microcomputers	
– Digital I/O and timer	160.00
– Eight serial port card	N/A

6. Full Scale Lab Systems

Cyborg	
– ISAAC	
– 41A chassis	1,600.00
– 91A system chassis	4,000.00
– 850-130 2 by I slot chassis	150.00
– 850-129 6 by I slot chassis	700.00
– 2000 system chassis	7,925.00
– 850-161 6 by C slot chassis	275.00
– I-140A analog input conditioner	800.00
– I-140B analog input conditioner	750.00
– I-140C analog input conditioner	750.00
– I-100 A/D converter	850.00
– I-150 A/D converter	1,100.00
– I-180 A/D converter	700.00
– I-130 A/D converter	600.00
– C-100 A/D converter	3,825.00
– C-200 256K RAM	2,100.00
– I-110 analog output	750.00
– I-120 digital I/O	350.00
Eco-Tech, Inc.	
– ALIS A12 Precision Analog input	1,517.00
– ALIS A08 Analog input	1,149.00
– ALIS AO Anaolog output	841.00
– ALIS DIO Digital input/output	1,600.00
Interactive Microware	
– Adalab	495.00
– Ada-Mux	195.00
– Ada-Byte (32 bit digital multiplexer)	395.00
– Ada-Amp w/program gain and multiplexer	300.00
– Chromadapt	350.00
Interactive Structures	
– DAISI (Data Acquisition and Instrumentation Systems Interface	
– AI13 (12 bit analog input)	550.00
– AI02 (8 bit analog input)	299.00
– AO03 (8 bit analog output) 2 channel	195.00
4 channel	275.00
8 channel	440.00
– DI09 (Digital timer interface)	330.00
– SC-14 Signal conditioners 1 channel	44.00
4 channel	275.00
16 channel	440.00
– UI-16 PB power control 4 channel	95.00
8 channel	120.00
16 channel	155.00
– UI-16 PAM power control	395.00

Keithley/DAS	
– Chassis w/software	2,100.00
– AIM1 input signal conditioner	850.00
– AIM2 input signal conditioner	300.00
– AIM3 input signal conditioner	575.00
– AIM4 input signal conditioner	760.00
– AIM5 input signal conditioner	780.00
– AIM6 input signal conditioner	680.00
– ADM1 A/D Converter	700.00
– ADM2 A/D Converter	950.00
– AOM1 Analog output, 2 channel	500.00
– AOM1 Analog output, 5 channel	800.00
– AOM2 Analog output, 1 channel	700.00
– AOM2 Analog output, 2 channel	1,400.00
– AOM3 Analog output	N/A
– DIM1 digital input	250.00
– DOM1 digital output	250.00
– PCM1 power control	350.00
– PCM2 power control	500.00

7. Specialized Input and/or Control Systems

Micro General Corporation	
– Weighmate (scale)	695.00

8. Robots

Androbot	
– Topo	1,595.00
Colne Robotics	
– Armdroid 1	N/A
RB Robot Corp.	
– RB5X Robot	1,795.00
Rhino Robots	
– XR-2	N/A
Zenith (Heath)	
– Hero I	2,500.00

F. Digital Ports

1. Modem Cards
2. External Modems
3. Serial Interface Cards
4. Parallel Interface Cards
5. Graphics Parallel Printer Cards
6. Printer Buffer Cards
7. External Printer Buffers
8. 1EEE-488
9. Network Equipment
10. Videotex

1. Modem Cards

Hayes Microcomputer Products	
– Micromodem II	380.00
– Micromodem //e	290.00
Microcom	
– ERA2 1200 baud modem card w/terminal emulation	430.00
Multi-Tech Systems Inc.	
– Modem II	400.00
– Multi-Modem II (300/1200)	N/A
Novation Inc.	
– Apple-Cat II	390.00
– Apple-Cat II 212 upgrade card	390.00
– 212 Apple-CAT II	725.00
– Expansion Module	40.00
– BSR module	20.00
– Handset	30.00
– Touch tone receiver	100.00
– Deaf Firmware	30.00
SSM Microcomputer Products Inc. (Transend)	
– Modemcard	325.00
Timecor	
– The Operator (modem card)	160.00
Zoom Telephonics	
– Networker 300 baud direct connect	130.00

2. External Modems (require serial card)

Anchor Automation	
– Signalman Mark I 300 baud	100.00
– Volksmodem (battery powered)	70.00
– Signalman Mark VII autodial/auto answer 300 baud	160.00
– Signalman Mark XII autodial/auto answer 1200 bps	400.00
Apple Computer	
– Apple Modem 300	225.00
– Apple Modem 300/1200	495.00
Hayes Microcomputer Products	
– Smartmodem 300	290.00
– Smartmodem 1200	700.00
M & R Enterprises	
– Sup'R Access-1, 300 baud/Bell 202, w/4 ports	700.00
Multi-Tech Systems Inc.	
– MT212 AD	700.00
– MT103	N/A
– FM30 acoustic couplers	N/A
– MT202T 1200 bps	N/A

– MT25 break out/monitor	N/A
– MT1001D direct connect data coupler	N/A
– Mulit-Mux 402	N/A
Novation Inc.	
– Cat (acoustic modem)	190.00
– D-Cat (direct connect)	200.00
– J-Cat originate/autoanswer miniature	150.00
– 212 Auto-Cat autodial	700.00
– 103 Smart-Cat	250.00
– 103/212 Smart-Cat	600.00
Prometheus	
– Pro Modem 1200	500.00
SSM Microcomputer Products Inc. (Transend)	
– Transmodem 1200	700.00
U.S. Robotics	
– Password 1200 baud modem	450.00
USI	
– Autodial/autoanswer modem external 300 baud	100.00

3. Serial Interface Cards (for Modems and Printers)

Advanced Logic Systems	
– Dispatcher serial	140.00
Apple Computer	
– Communications Board	N/A
– Super Serial	225.00
– Serial Interface Card	N/A
California Computer Systems	
– Model 7710A Asynchronous Serial (for printer)	135.00
– Model 7710D Asynchronous Serial (for modem)	135.00
– Model 7711 Asynchronous Serial (modem or printer)	110.00
CTA (Computer Technology Associates)	
– Generic Serial Communication Card	110.00
FM Panatronics	
– Programmable Serial Card	150.00
M & R Enterprises	
– Adaptabox	50.00
– Sup'R Access-1, 4 switch selectable RS-232C ports	700.00
MBI	
– VIP Graphics Card w/serial port	120.00
MPC Peripherals	
– AP-SIO, serial card, includes Imagewriter driver	169.50
Microtek	
– SV-622 Serial Interface Card	100.00

Mountain Computer	
– CPS Multifunction (out of production)	240.00
PC Ware, Inc.	
– Serial Interface	130.00
– Enhanced Serial	185.00
Practical Peripherals	
– Microbuffer II 16K	200.00
– Microbuffer II 32K	220.00
– Microbuffer II+ (Ser/Par/Graphics)	
– 16K	260.00
– 32K	300.00
– 64K	350.00
– SeriALL w/Imagewriter Graphics ROM	160.00
Prometheus	
– P/S 16K buffer	199.00
– Versacard (Serial, Parallel, Clock, BSR Control)	200.00
SSM Microcomputer Products Inc. (Transend)	
– ASIO	150.00
– AIO-II	225.00
U-Microcomputers	
– Eight serial port card	N/A
Videx	
– PSIO	230.00

4. Parallel Interface Cards

Advanced Logic Systems	
– Printermate Centronics Parallel	100.00
Apple Computer	
– Parallel Interface	165.00
California Computer Systems	
– Model 7720 Parallel	100.00
– Model 7728 Centronics Parallel Interface	100.00
– Model 7731 Centronics Parallel	70.00
Coex (Components Express)	
– Parallel Card	100.00
MPC Peripherals	
– AP-80 Parallel Printer Card	130.00
Micromax	
– PrintMax	90.00
Microtek	
– RV-611C Parallel Interface 7 or 8 bit parallel	140.00
Mountain Computer	
– CPS Multifunction (out of production)	240.00

Orange Micro	
– Orange Printer Interface	90.00
PC Ware, Inc.	
– General Parallel	90.00
– Centronics Parallel	80.00
Practical Peripherals	
– Microbuffer II 16K	200.00
– Microbuffer II 32K	220.00
– PRinterface	N/A
Prometheus	
– Parallel Interface Card	100.00
– Versacard (Serial, Parallel, Clock, BSR Control)	200.00
Quadram	
– APIC (Parallel Interface Card)	140.00
SSM Microcomputer Products Inc. (Transend)	
– AIO-II	225.00
– APPIC/G Graph-It parallel	110.00
U-Microcomputers	
– U-Print	200.00
– U-Centronics	80.00
Videx	
– PSIO	230.00
– Uniprint	90.00

5. Graphics Parallel Printer Cards

Interactive Structures	
– Pkaso EP12-80	175.00
– Pkaso EP12-100	175.00
MBI	
– VIP Graphics Card w/serial port	120.00
MPC Peripherals	
– Graphics parallel	175.00
– Graphics parallel w/64K buffer	370.00
Macrotech Computer Products Ltd.	
– Macroprint parallel card w/zoom	180.00
Micromax	
– GraphMax	150.00
– GraphMax w/color and zoom	170.00
Microtek	
– Dumpling-GX (2K buffer)	90.00
– Dumpling-16	215.00
– Dumpling-32	240.00
– Dumpling-64	285.00

Orange Micro	
– The Grappler +	175.00
– Buffered Grappler (64K)	240.00
Practical Peripherals	
– Microbuffer II+ (Ser/Par/Graphics)	
– 16K	260.00
– 32K	300.00
– 64K	350.00
– GraphiCard	N/A
Prometheus	
– Graphitti Graphics parallel	130.00
Quadram	
– APIC/G (Parallel Interface Card)	160.00
Speis Laboratories	
– Super MX Wordstar/dot matrix interface	175.00
Tex Print Inc.	
– Print It (screen grab, NMI)	300.00
– Model 2	175.00
Thirdware Computer Products	
– Fingerprint (screen grab, NMI)	150.00

6. Printer Buffer Cards

MPC Peripherals	
– 64K Buffer card	300.00
– Graphics parallel port w/64K buffer	370.00
Microtek	
– Dumpling-16 (buffered parallel card)	250.00
– Dumpling-32 (buffered parallel card)	300.00
– Dumpling-64 (buffered parallel card)	400.00
Orange Micro	
– Bufferboard	150.00
– Buffered Grappler, parallel port w/64K	240.00
Practical Peripherals	
– Microbuffer II 16K (w/parallel or serial port)	200.00
– Microbuffer II 32K (w/parallel or serial port)	220.00
– Microbuffer II+ (serial/parallel/graphics)	N/A
– 16K	260.00
– 32K	300.00
– 64K	350.00
Prometheus	
– P/S 16K buffer	199.00
Wesper Microsystems	
– BPO 16K parallel	180.00
– BPO 32K parallel	220.00
– SOB 16K serial	250.00
– SOB 32K serial	290.00

7. External Printer Buffers

Interactive Structures	
– Pipeline 64K buffer	230.00
– Shuffle buffer	300.00
MPC Peripherals	
– Omnibuffer	400.00
Microtek	
– Bufferbox 64K	400.00
– Bufferbox 128K	500.00
Practical Peripherals	
– Microbuffer in-line serial or parallel 32K – 256K	
– 32K	300.00
– 64K	350.00
– 64K add on modules for Microbuffer in-line	180.00
– Snapshot option	120.00
Quadram	
– Microfazer buffer	N/A
– 64K	300.00
– 256K	900.00
– 512K	1,400.00

8. IEEE-488

Apple Computer	
– IEEE-488	450.00
California Computer Systems	
– Model 7490 IEEE-488 Interface	200.00
SSM Microcomputer Products Inc. (Transend)	
– IEEE-488 Controller	475.00

9. Network Equipment

Apple Computer	
– Applebus interface	N/A
– 3270 Cluster Controller, 3 ports	4,500.00
– 3270 Cluster Controller, 7 ports	7,000.00
– Apple Line to existing 3270 controller	1,300.00
– Protocol Card 3270 terminal emulation	700.00
California Computer Systems	
– Model 7712 Synchronous Serial Interface	175.00
Corvus	
– Omninet	N/A
Davong	
– Network	N/A

Enlink Inc.	
– Ethernet interface	N/A
Nestar	
– PLAN 3000	10,000.00
U-Microcomputers	
– U-Net network controller	N/A

10. Videotex

Norpak	
– Telidon Graphics	600.00

G. Printers and Plotters

1. Dot Matrix Printers
2. Ink Jet Printers
3. Laser Printers
4. Letter Quality Printers
5. Plotters

1. Dot Matrix Printers

Apple Computer	
– Imagewriter, 180 cps, serial port only	595.00
– Dot Matrix Printer, 120 cps	N/A
– Silentype (thermal printer)	350.00
Axiom (Seikosha)	
– EX-1620 240-960 cps	795.00
– TX-1000 video printer (uses video signal)	3,400.00
– IMP-40 100 cps	N/A
– GP-100A 30 cps	300.00
– GP-250X 50 cps dot matrix	340.00
– color printer	600.00
Epson	
– MX-80, 80 cps	525.00
– MX-100, 80 cps	750.00
– RX-80, 60 cps	500.00
– FX-80, 160 cps	700.00
– FX-100, 160 cps	900.00
Integral Data Systems	
– Prism 480, 100 cps	650.00
– P-Series 80	1,300.00
– P-Series 132	1,500.00
– color	+400.00

Leading Edge (C.Itoh)	
– Prowriter 8510A, 120 cps	595.00
– Prowriter 2 Printer-1550, wide carriage, 120 cps	995.00
– Gorilla (Seikosha GP-100A), 30 cps	250.00
Mannesman Tally	
– MT 160L, 160 cps	770.00
– Spirit	400.00
NEC Home Electronics	
– 8923A, dot matrix printer, 100 cps	645.00
– 8025A, dot matrix printer, 120 cps	975.00
Okidata Corp.	
– Microline 82, 120 cps, 80 column	500.00
– Microline 84, 200 cps, 136 column	1,400.00
– Microline 92, 160 cps, 80 column	600.00
– Microline 93, 160 cps, 136 column	1,000.00
– Pacemark 2410, 350 cps	3,000.00
Star Micronics	
– Gemini 10X, 120 cps, 80 col.	400.00
– Gemini 15X, 120 cps, 132 col.	650.00
– Delta 10, 160 cps	650.00
– STX-80, 60 cps	200.00
– Radix-15, 200 cps	N/A
Toshiba America, Inc.	
– P1350, 190 cps dot/letter quality	2,195.00
– P1340, 100 cps dot/letter quality	995.00
Transtar (Seikosha)	
– Transtar 315 color dot matrix	600.00

2. Ink Jet Printers

Advanced Color Technology	
– ACT-1 Color printer	N/A
– ACT-2 Color printer	6,150.00
– video interface	2,345.00
– serial interface	260.00
Canon	
– PJ-1080A Color Printer, 37 cps 640 dots/line	795.00
Diablo	
– Series C, 20 cps	1,295.00
Hewlett Packard	
– Think Jet	495.00
Quadram	
– Quadjet color printer, 640 dots/line, 35 cps	895.00
Siemens Corp.	
– PT-88, 150 cps	895.00
– 2712, 270 cps	2,250.00

3. Laser Printers

Apple Computer	
– Laser Printer	5,000.00
Canon	
– LBP-CX, 120 cps	3,000.00

4. Letter Quality Printers

Apple Computer	
– Letter Quality Printer	2,195.00
Comrex International, Inc.	
– ComRiter CR-I, 17 cps (Qume Sprint 3)	800.00
– ComRiter CR-II, 12 cps	600.00
– ComRiter CR-III, 23 cps	1,000.00
DTC (Data Terminals and Communication)	
– 380Z, 32 cps (Diablo code)	1,495.00
Diablo	
– Model 630, 40 cps	1,995.00
– Model 620, 20 cps	1,050.00
Juki Industries of America, Inc.	
– Model 6100 daisy wheel printer, 18 cps	700.00
Leading Edge (TEC) (C.Itoh)	
– Starwriter F10-40	1,895.00
– Printmaster F10-55	2,000.00
NEC Home Electronics	
– 15LQ, letter quality printer	700.00
NEC Information Systems	
– Series 2000 letter quality printers, 20 cps	1,100.00
– Series 3500 letter quality printers, 33 cps	2,200.00
– Series 7700 letter quality printers, 55 cps	2,600.00
Qume	
– Sprint 11/44 Plus, 40 cps	1,685.00
– Sprint 11/55 Plus, 55 cps	1,990.00
– Sprint 11/40-130, 130 character print wheel	2,965.00
– Letter Pro 20	N/A
Star Micronics	
– Powertype letter quality, 18 cps	N/A
Transtar (Seikosha)	
– Transtar 120 (portable daisy wheel) (14cps, Diablo 1610 code compatible)	600.00
– Transtar 130 18 cps, 1610 compatible	900.00
– Transtar 140 40 cps, 1610 code compatible	1,700.00

5. Plotters

Amdek	
– Amplot-II digital plotter, 6 pen	1,099.00
– DXY-100 plotter	750.00
Apple Computer	
– Color Plotter Model 410, 4 pen	995.00
Bausch and Lomb (Houston Instruments)	
– Hiplot DMP-29, 8 pen, 11 by 17	2,295.00
– Hiplot DMP-40	995.00
– DMP-41	3,000.00
– D-5000 Omniscribe analog strip chart recorder	1,000.00
– 4500 Microscribe smart anlaog strip chart	835.00
– Complot CPS-19	14,000.00
– Complot CPS-20	4,000.00
– Complot CPS-30	6,000.00
Comrex International, Inc.	
– ComScriber I	795.00
Enter Computer, Inc.	
– Sweet-P plotter	800.00
– Six-Shooter, 6 pen, 11 by 17	1,095.00
Hewlett-Packard	
– 7470 2 pen plotter	1,095.00
– 7475 6 pen plotter	1,895.00
Strobe Inc.	
– M100 single pen drum plotter	595.00
– M260 6 pen plotter	995.00

H. Memory Boards

1. 16K RAM Cards for II/II+
2. //e 80 Column, 64K RAM
3. High Capacity RAM cards for //e
4. High Capacity Bank Switch RAM cards
5. Memory w/non-standard access
6. ROM Equipment

1. 16K RAM Cards for II/II+

Advanced Logic Systems	
– Add-RAM 16K card	100.00
Apple Computer	
– Language Card	100.00
Coex (Components Express)	
– 16K RAM	100.00

ComX	
– 16K Ramcard II+	180.00
Legend Industries	
– 18K SRC static RAM w/battery back up	160.00
MPC Peripherals	
– 16K board	149.00
Macrotech Computer Products Ltd.	
– Macromem-1 16K	100.00
– Macromem-2 32K	180.00
Microsoft	
– 16K card	100.00
Microtek	
– Bam-16, 16K card	100.00
Mountain Computer	
– Ramplus+ (32K)	220.00
Omega Microware	
– Ramex 16K	140.00
Prometheus	
– MEM-1 16K	100.00
STB Systems Inc.	
– STB 16K	170.00
Saturn Systems Inc. (Titan Technologies)	
– 32K Ram Board	220.00
Wesper Microsystems	
– Wizard-16K	95.00

2. //e 80 Column, 64K RAM

Apple Computer	
– //e 64K extended text card	295.00
Applied Engineering	
– Memory Master //e (64K)	170.00
Coex (Components Express)	
– //e Extended RAM card	200.00
ComX	
– //e 64K, 80 Col.	300.00
MPC Peripherals	
– Ramtext //e (64K)	290.00
Micromax	
– Viewmax-80e (64K)	190.00
Microsoft	
– Premium SoftCard //e Z-80B w/64K RAM	495.00

Microtek	
– Magnum 80Me (64K)	140.00
PC Ware, Inc.	
– Extended //e 80 col (64K)	160.00
Quadram	
– eRAM 80 (//e 64K and 80 column)	160.00
Saturn Systems Inc. (Titan Technologies)	
– Neptune Board (//e)	
– 64K	250.00
Taxan	
– Model 410-64 (64K, w/RGB interface)	350.00

3. High Capacity RAM Cards for the //e

Applied Engineering	
– Memory Master //e (128K)	250.00
Micromax	
– Viewmax-80e (128K)	300.00
Saturn Systems Inc. (Titan Technologies)	
– Neptune Board (//e)	
– 128K	400.00
– 192K	500.00

4. High Capacity Bank Switch RAM Cards

Abacus	
– Know-Drive 128K w/NMI playback option	485.00
Legend Industries	
– 64K KC	327.00
– 128K DE	600.00
– S'Card 64K to 1 megabyte (w/256K chips)	
– 64K	400.00
– 128K	525.00
– 192K	624.00
– 256K	725.00
Macrotech Computer Products Ltd.	
– Macromem-3 64K	300.00
128K	450.00
– Diskulator	
– 64K	450.00
– 256K	550.00
– 512K	1,100.00
Omega Microware	
– Ramex 128K	500.00

Prometheus	
– Expand-a-RAM	
– 64K	375.00
– 128K	495.00
STB Systems Inc.	
– STB 16/64K	250.00
– STB 64K	N/A
– STB 128K	400.00
Saturn Systems Inc. (Titan Technologies)	
– 64K Ram Board	350.00
– 128K Ram Board	500.00
U-Microcomputers	
– U-RAM 128K	450.00

5. Memory w/non-Standard Access

Axlon	
– Ramdisk 320 (external, battery backup)	1,000.00
– Ramdisk 128 card	380.00
Cramapple	
– Adapter for 64K RAM chips in II/II+ (remove motherboard 48K, install 192K)	N/A
– no-mod adapter	120.00
– adapter requiring motherboard modification	80.00
– 24, 64K bit RAM chips for 192K total	250.00
– Adapter for 256K RAM chips in //e	120.00
MPC Peripherals	
– Bubdisk 128K bubble memory	875.00
Microtek	
– BAM 64K	200.00
– BAM 128K DE	300.00
– Q-Disk, 128K RAM	400.00
Personal Computer Products Inc.	
– Appli-Card Z-80B, 6 MHz	375.00
– 64K RAM extender	195.00
– 128K RAM extender	295.00
Pion, Inc.	
– Interstellar Drive (battery back 256K; 1 megabyte)	
– 256K	1,095.00
– 256K expansions	595.00
Synetix	
– SSD Solid State Disk Emulator 144K	350.00
– SSD Solid State Disk Emulator 288K	530.00

6. ROM Equipment

Apple Computer	
– ROM card	N/A
California Computer Systems	
– Model 7114 12k ROM/PROM module	115.00
CTA	
– Apple Prom EPROM burner	150.00
Hollister Micro Systems	
– HMS 3264 EPROM programmer	395.00
Hollywood Hardware	
– Ultra-ROM board	190.00
Mountain Computer	
– ROMplus+	155.00
– ROM Writer	175.00

I. Magnetic Storage

1. Disk Controller Cards
2. Standard Disk II Compatible Drives
3. Thinline Disk II Compatible Drives
4. High Capacity Floppy Drives
5. Floppy Drive Accessories
6. Fixed Hard Disk Drives
7. Removable Hard Disks
8. Hard Disk Backup
9. Nine Track Magnetic Tape

1. Disk Controller Cards

Apple Computer	
– Disk II Controller	N/A
– Duodisk Controller	N/A
– Profile Controller	N/A
– 3 1/2 inch Controller	N/A
ALF	
– DC3 Apple/IBM disk controller	350.00
Data Cue, Inc.	
– Disc Master II, 8 inch or 3 1/2 inch MFM controller	265.00
Micro Sci	
– Disk II compatible controller	100.00
Prometheus	
– Applesurance, disk controller and diagnostics	125.00
Quadram	
– Quadlink w/controller for IBM PC drives	680.00

Rana	
– 8086/2 (IBM PC and Disk II compatible)	1,795.00
– Four drive controller card	100.00
Sorrento Valley Associates	
– ZVX4 8 inch controller (out of production)	445.00
– Megaflex controller (out of production)	300.00
Vista	
– A800 Controller	379.00

2. Standard Disk II Compatible Drives

Apple Computer	
– Disk II	395.00
Lobo	
– Apple compatible drive	385.00
Micro Sci	
– A 2 Disk Drive (143K)	345.00
– XL-drive	200.00
Mitac	
– Mate-I Apple compatible	170.00
Quentin Research	
– AP-100	335.00
Rana	
– Elite One (163K Drive)	380.00
Vista	
– Solo (Disk II compatible)	300.00

3. Thinline Disk II Compatible Drives

Apple Computer	
– Duodisk, dual thinline	800.00
Comrex International, Inc.	
– ComDrive CR-1000, dual thinline	600.00
Quentin Research	
– AP-105	300.00
– AP-110 dual thinline w/controller	600.00

4. High Capacity Floppy Drives

Amdek	
– Amdisk-III 3 inch drive, 143K per side	300.00
Apple Computer	
– 3 1/2 inch drive 400K	495.00

Eicon Research Inc.	
– Tera-Drive 2 megs in two thinline 5 1/4 inch floppies	N/A
Lobo	
– 8 inch drives	1,925.00
Micro Sci	
– A 40 Disk Drive (160K)	380.00
– A 70 Disk Drive (280K)	530.00
– A 82 Disk Drive (320K)	N/A
Rana	
– 8086/2 (IBM PC and Disk II compatible)	1,795.00
– Elite One (163K drive)	380.00
– Elite Two (320K drive)	650.00
– Elite Three (640K drive)	850.00
– 2.5 Megabyte Floppy	1,500.00
Taurus Computer Systems, Inc.	
– 8-inch controller and single drive	888.00
– 8-inch controller and dual drive	1,488.00
Vista	
– A800 + dual SSDD 8 inch, V1000 cabinet (1.2 meg)	1,500.00
– A800 + dual DSDD 8 inch, V1000 cabinet (2.4 meg)	1,800.00
– A800 + dual DSDD slimline 8 inch, V1100 cabinet	2,100.00
– A801 + V1200 6MB floppy	1,500.00
– Duet, thinline 5 1/4 inch (320K)	425.00
– Quartet (two by 320K)	850.00

5. Floppy Drive Accessories

Mountain Computer	
– Diskette quantity duplications systems	
– 3 1/2 inch	N/A
– 5 1/4 inch	N/A
– 8 inch	N/A
Verbatim	
– Disk Drive Analyzer	40.00

6. Fixed Hard Disk Drives

Apple Computer	
– ProFile, 5 megabytes	1,495.00
– 70 megabytes	7,000.00
Axlon	
– 5MB Hard Disk	1,200.00
– 5MB fixed in cabinet w/5 MB removable	1,900.00
Corona Data Systems	
– Starfire Hard Disk 5 megabyte	2,195.00
– 10 megabyte	2,695.00

Corvus	
– Interface card	300.00
– Hard Disk 6 AP	2,095.00
– Hard Disk 11 AP 20 megabytes	2,750.00 3,750.00
Davong	
– Hard Disk	
– 5 megabyte	1,995.00
– 10 megabyte	2,400.00
– 15 megabyte	2,800.00
– 21 megabyte	3,300.00
– 32 megabyte	4,000.00
Eicon Research Inc.	
– DisCache Hard Disk (w/RAM for high speed anticipatory track buffering, network server, incremental backup)	
– 10 megabyte w/128K buffer	3,350.00
– 20 megabyte w/256K buffer	4,250.00
– 40 megabyte	4,900.00
Lobo	
– 5 megabytes hard disk w/floppy	2,675.00
Mitac	
– 10 MB hard disk	1,500.00
Mountain Computer	
– Dynamic Hard Disk System	
– 5 megabyte	1,995.00
– 10 megabyte	2,495.00
– 15 megabyte	2,995.00
– 20 megabyte	3,495.00
Percom Data Corp.	
– Hard Disk	
– 5 megabyte	1,900.00
– 10 megabyte	2,300.00
– 15 megabyte	2,800.00
– 20 megabyte	3,300.00
Quentin Research	
– Q-500	
– 5 megabytes	N/A
– 10 megabyte	2,000.00
– 15 megabyte	2,300.00
VR Data	
– Dual 10 megabyte hard disk	2,000.00
Xebec	
– Hard Disk 5 megabyte kit	1,300.00

7. Removable Hard Disks

Axlon	
– Removable 5 megabyte w/5 megabyte fixed	1,900.00

Digital Electronics Systems
- Removable Hard Disk 1,300.00

VR Data
- Removable Hard Disk 5 megabyte 1,495.00

Vufax, Inc.
- Infax 101A 10 megabyte Hard Disk cartridge 2,400.00

X-Comp, Inc.
- The Toaster (2 by 5 megabyte, removable) 2,795.00

8. Hard Disk Backup

Corvus
- The Bank 200 megabyte backup tape 2,200.00

Davong
- Tape Backup 1,995.00

Mountain Computer
- FileSafe Tape Backup System (20 megabyte) 2,195.00

Vista
- A801 + V1200 6MB floppy 1,500.00

9. Nine Track Magnetic Tape

Electrovalue Industrial, Inc.
- Model 1025 full size magnetic tape, 800 bpi 3,000.00
- Pertec 7000 7 inch tape 1,800.00

Innovative Data Technology
- IDT-TD1050 system 8,500.00

J. Co-processors

1. 6502 Based Products
2. 6809
3. Z-80A
4. Z-80B
5. Intel 8088/8086/8087
6. Motorola 68000
7. WDC 65816

1. 6502 Based Products

Abacus Compusource
- Portable Apple compatible computer 1,795.00

Applied Analytics	
– Microspeed II (2 MHz AMD 9511)	495.00
– Microspeed II+ (4 Mhz AMD 9511)	645.00
– Booster 4 MHz 6502C	600.00
California Computer Systems	
– Arithmetic Processor (2 MHz AMD9511)	360.00
Micro Computer Technologies	
– Speedemon 6502C without RAM	295.00
MicroSci	
– HAVAC slotless 64K Apple II+ compatible	850.00
Saturn Systems Inc. (Titan Technologies)	
– Accelerator	600.00
Southwestern Data Systems	
– 65C02 Upgrade	35.00

2. 6809

Norpak	
– Telidon Graphics Videotex Processor	600.00
Stellation Two	
– The Mill 6809	295.00
– VitaMill w/64K RAM	450.00

3. Z-80A

Advanced Logic Systems	
– Z-card II	170.00
Applied Engineering	
– Z-80 Plus	140.00
Excalibur Technologies	
– Savvy Language System	
– Savvy One	350.00
– Savvy Pro	500.00
– Business Savvy	950.00
Micromax	
– Z-Max, 4 MHz Z-80A	160.00
Microsoft	
– SoftCard	345.00
– SoftCard //e	345.00
Personal Computer Products Inc.	
– Appli-Card Z-80A, 4 MHz	295.00
– 64K RAM extender	195.00
– 128K RAM extender	295.00
U-Microcomputers	
– U-Z80	140.00

4. Z-80B

Advanced Logic Systems	
– CP/M Card Z-80B w/64K RAM	400.00
Digital Research	
– Gold Card with CP/M 3.0	325.00
– w/64K	495.00
– w/128K	775.00
Microsoft	
– Premium SoftCard //e Z-80B w/64K RAM	495.00
Personal Computer Products Inc.	
– Appli-Card Z-80B, 6 MHz w/64K RAM	375.00
– 64K RAM extender	195.00
– 128K RAM extender	295.00

5. Intel 8088/8086/8087

ALF	
– DC3 Apple/IBM disk controller	350.00
– AD8088 Processor card with FTL and MET	345.00
– Expansion Board for AD8088 (sockets for 64/128K)	295.00
– 64K	75.00
– 8087	262.50
Abacus Compusource	
– Portable Apple compatible computer	1,795.00
– PC Mate upgrade for PC compatibility	N/A
Metamorphic Systems Inc.	
– MetaCard	
– 64K	850.00
– 128K	980.00
Personal Computer Products Inc.	
– Co-processor 8088 64K only	595.00
Rana	
– 8086/2 (IBM PC and Disk II compatible)	
– 256 K, two thinline 360K drives,	N/A
– 640 by 400 hi-res graphics	1,795.00
– w/512K	N/A

6. Motorola 68000

Acorn	
– 68000 board	1,600.00
Analytical Engines	
– Saybrook 68000, 128K	
– 8 MHz	900.00
– 12.5 MHz	1,200.00
– w/512K RAM	N/A
– 14 MHz	1,400.00

Digital Acoustics	
– DTACK Grande (68000)	
– 128K	795.00
– 1 megabyte	1,995.00
– DTACK Grounded (68000)	
– 4K high speed RAM	683.00
– 92K	1,123.00
ETC (Enhancement Technologies)	
– PDQ II (68000) board w/128K RAM	1,495.00
IBS Computertechnik	
– AP20 Intemex w/68000 CPU and 128K	650.00
– additional 256K of RAM	720.00
Qwerty, Inc.	
– Q-Pak 68 development system, no RAM 68008 uses Apple motherboard memory	695.00
Stellation Two	
– 68008 co-processor	230.00

7. WDC 65816

Western Design Center	
– 65816 tool box	1,495.00
– 65816 based CAD chip design system	5,000.00

K. NMI Boards

1. Foreground/Background
2. Program Copiers
3. Screen Grab Printer Card

1. Foreground/Background

Abacus	
– Know-Drive 128K w/NMI playback option	485.00

2. Program Copiers

Dark Star Systems	
– Snapshot //e	140.00
– Snapshot II	120.00
East Side Software	
– Wildcard	110.00
– Wildcard 2 for 64K or 128K //e	140.00
– Wildcard Plus (has 6502 on board)	170.00

Micro-Analyst Inc.
- Replay II 80.00

Practical Peripherals
- Snapshot Option 120.00

3. Screen Grab Printer Card

TexPrint
- Print It 300.00

Thirdware Computer Products
- FingerPrint 150.00

L. Utility Software

1. Assemblers
2. Ampersand Routines
3. Applesoft Programmer's Aids
4. Applesoft Compilers

1. Assemblers

Apple Computer
- ProDOS Toolkit N/A

Hayden Software
- ORCA/M 150.00

Lazerware
- LISA 80.00

Microsoft
- ALDS 125.00

Southwestern Data Systems (Roger Wagner Publishing)
- Merlin 65.00

2. Ampersand Routines

Anthro-Digital
- Amper-Magic 75.00
- Amper-Magic Command Library 35.00

MicroSparc
- Ampersoft 50.00

Southwestern Data Systems (Roger Wagner Publishing)
- Routine Machine 65.00

3. Applesoft Programmer's Aids

ALF
- HGR6 //e super Hi-Res routines 50.00

Beagle Brothers
- GPLE w/DOS mover with PEEKs and POKEs chart 50.00

Southwestern Data Systems (Roger Wagner Publishing)
- Apple-DOC 40.00
- Listmaster w/Applespeed 40.00

4. Applesoft Compilers

Einstein Corp.
- Expediter II 150.00

Hayden
- Applesoft Compiler 60.00

Microsoft
- TASC 175.00

Southwestern Data Systems (Roger Wagner Publishing)
- Speedstar 135.00

M. Reading

1. Magazines
2. Hardware Reference
3. Assembly Language Programming Reference
4. Applesoft, DOS and CP/M Programming Reference

1. Magazines

A+
- general readership; one year subscription 25.00

Apple Orchard
- general readership; one year subscription 24.00

Call A.P.P.L.E.
- programmers; one year subscription 25.00

In Cider
- programmmers; one year subscription 25.00

Nibble
- programmers; one year subscription 25.00

Softalk
- general and programmers; one year subscription 24.00

2. Hardware Reference

Apple II User's Guide by Lon Poole (Osborne/McGraw Hill)
– for beginners

Apple II Circuit Description by Wiston Gayler (Sams)
– very technical

Introduction to Microcomputers: Volume 1 by Adam Osborne (Osborne/McGraw Hill)
– intermediate and advanced

Understanding the Apple II by Jim Sather (Quality Software)
– very technical

3. Assembly Language Programming Reference

Apple Almanac by Eric Goez and William Sanders (DATAMOST)
– advanced

Assembly Lines by Roger Wagner (Softalk)
– beginning

Assembly Language Programming for the Apple II by Robert Mottola (Osborne/McGraw Hill)
– beginning

6502 Assembly Language Programming by Lance Leventhal (Osborne/McGraw Hill)
– advanced

Programming the 6502 by Rodney Zaks (Sybex)
– intermediate and advanced

Using 6502 Assembly Language by Randy Hyde (DATAMOST)
– intermediate

What's Where in the Apple by William Luebbert (Micro Ink)
– advanced

4. Applesoft, DOS and CP/M Programming Reference

Apple BASIC: Data File Programming by LeRoy Finkel and Jerald Brown (Wiley)
– intermediate

Applesoft Encyclopedia by Loy Spurlock (DATAMOST)
– intermediate and advanced

Beneath Apple DOS by Don Worth and Pieter Lechner (Quality Software)
– advanced

Elementary Apple by William Sanders (DATAMOST)
– beginners and intermediate

Inside CP/M by David Cortesi (Holt, Rinehart, and Winston)
– intermediate and advanced

Kids and the Apple by Edward Carlson (DATAMOST)
– beginners

Programming the Apple by J.L. Campbell and Lance Zimmerman (Brady)
– intermediate

Appendix H
Dealer List

Western Region

ALASKA

Anchorage
Computerland, Anchorage
502 W. Northern Lights Blvd.
Anchorage, AK 99503
(907) 276-5941

Empire Electronics
5011 Arctic, Ste. B
Anchorage, AK 99502
(907) 279-7916

Team Electronics #031
404 E. Fireweed Ln.
Anchorage, AK 99503
(907) 272-4823

Fairbanks
Team Electronics #142
1698 Airport Way
Fairbanks, AK 99701
(907) 456-4157

Juneau
Juneau Electronics
1000 Harbor Way
Juneau, AK 99801
(907) 586-2260

Soldotna
Computer Concepts
Box 4128
Soldotna, AK 99669
(907) 262-5612

ARIZONA

Flagstaff
The Computer Bank
2636 N. Steves
Flagstaff, AZ 86001
(602) 526-6705

Mesa
Computerland, Tempe-Mesa
1310 W. Southern, Ste. 4
Mesa, AZ 85202
(602) 962-6732

Mesa Computer Mart
11 S. Morris St.
Mesa, AZ 85202
(602) 833-1155

Phoenix
The Computer Store
12416 N. 28th Dr., #20
Phoenix, AZ 85029
(602) 866-0258

Computerland
3152 E. Camelback Rd.
Phoenix, AZ 85016
(602) 956-5727

Computerland of Metro Center
Metro Center Pkwy.
Phoenix, AZ 85021
(602) 861-1394

Data Place Computer Store
3424 N. Central Ave., #110
Phoenix, AZ 85012
(602) 266-6111

Microage Computer Store
24 W. Camelback Rd.
Phoenix, AZ 85017
(602) 265-0065

Wabash Computer Systems
6102 N. 16th St.
Phoenix, AZ 85006
(602) 264-0546

Prescott
The Computer Room
115 W. Goodwin
Prescott, AZ 86301

Safford
Interstate Computers
Mt. Graham Spring Ctr.
Safford, AZ 85546
(602) 428-3357

Scottsdale
Computer Pro
2302 N. Scottsdale Rd.
Scottsdale, AZ 85257

Computer Systems BFA
4231 Winfield Scott Plz.
Scottsdale, AZ 85251
(602) 994-8792

Tempe
Dataplace Computer Stores
3136 S. McClintock Rd., Ste. 3
Tempe, AZ 85282
(602) 839-0888

Tucson
Computer Systems BFA
3955 E. Speedway Blvd.
Tucson, AZ 85712
(602) 881-8903

Computerland
6177 E. Broadway
Tucson, AZ 85711
(602) 790-8220

Simutek Computer Products
4897 E. Speedway
Tucson, AZ 85712
(602) 323-9391

Yuma
Data IV Office Supply
362 W. 32nd St.
Yuma, AZ 85012
(602) 344-4440

CALIFORNIA

Anaheim
Computerland of Anaheim
1500 N. State College Blvd.
Anaheim, CA 92806
(714) 935-3747

Net Profit Computers
521 W. Chapman Ave.
Anaheim, CA 92802
(714) 750-7318

Powers Computer Center
1295 N. Euclid
Anaheim, CA 92801
(714) 778-6021

Arcadia
Love Computers
7 E. Foothill Blvd.
Arcadia, CA 91006
(213) 447-0721

Bakersfield
Computer Basics
5600 California
Bakersfield, CA 93309
(805) 395-0802

The Computer Warehouse
2031 24th St.
Bakersfield, CA 93301
(805) 327-3393

Computerland of Bakersfield
3504 Ming Ave.
Bakersfield, CA 93309
(805) 832-5531

FMS Computers
4646 Wilson Rd., Ste. 103
Bakersfield, CA 93309
(805) 397-5864

Belmont
Computerland Belmont
1625 El Camino Real
Belmont, CA 94002
(415) 595-4232

Bishop
Bishop Typewriter
621 W. Line
Bishop, CA 93514
(714) 873-5773

Brea
Computer City
2700 E. Imperial Hwy.
Brea, CA 92621
(714) 996-0800

Computer Plus
805 W. Imperial Hwy.
Brea, CA 92621
(714) 990-3933

Computique
1080 E. Imperial Hwy.
Brea, CA 92621
(714) 990-6600

Buellton
The Computer Terminal
90 W. Hwy. 246
Buellton, CA 93427
(805) 688-1713

Burbank
Computerama Inc.
3808 W. Verdugo Ave.
Burbank, CA 91505
(213) 848-5521

Burlingame
Computerland of Burlingame
264 Lorton
Burlingame, CA 94010
(415) 348-7731

Capitola
Computer Age
1955 41st Ave., A-2
Capitola, CA 95010
(408) 476-6170

Carlsbad
Computerland San Diego, North
2502 El Camino Real
Carlsbad, CA 92008
(714) 434-7001

Carmel
Computer Place
26384 Carmel Rancho Ln.
Carmel, CA 93922
(408) 624-7111

Cerritos
Gateway Computer Center
11470 South St.
Cerritos, CA 90701
(213) 865-4444

Chico
Video Computer Systems
1600 Mangrove, Ste. H
Chico, CA 95926
(916) 891-1630

Citrus Heights
Computer Stores Inc.
6041 Greenback Ln.
Citrus Heights, CA 95610
(916) 969-2983

Concord
Computerland Concord
1701 Willow Pass Rd.
Concord, CA 94520
(415) 827-4985

Costa Mesa
Coast Computer Center
369 E. 17th St., #22
Costa Mesa, CA 92627
(714) 646-0537

May Co.
Major Appliances, 3333 Bristol St.
Costa Mesa, CA 92626
(714) 546-9321

Dana Point
The Computer Station
34207 Coast Hwy., #101
Dana Point, CA 92629
(714) 661-8062

Diamond Bar
Computermart of California
315 Diamond Bar Blvd., Ste. C
Diamond Bar, CA 91765
(714) 598-7505

Dublin
Computerland of Dublin
6743 Dublin Blvd.
Dublin, CA 94566
(415) 828-8090

El Cajon
Computer Store International
1251 Broadway
El Cajon, CA 92021
(619) 579-8066

El Cerrito
Computerland of El Cerrito
10042 San Pablo Ave.
El Cerrito, CA 94530
(415) 527-8844

El Centro
Office Supply Co.
561 Main St.
El Centro, CA 92243
(714) 352-3383

El Toro
The Wabash Apple
23720 El Toro Rd., Ste. C&D
El Toro, CA 92630
(714) 768-3610

Encinatas
Practical Computing
1472 Encinitas Blvd.
Encinatas, CA 92024
(619) 436-3512

Escondido
The Computer Merchant
1040 E. Valley Pkwy.
Escondido, CA 92027
(619) 746-0662

Fremont
Computerland of Fremont
3381 Walnut Ave., Crths Plz.
Fremont, CA 94538
(415) 794-9311

Fresno
Computerland of Fresno
4047 N. Blackstone Ave
Fresno, CA 93704
(209) 224-8200

National Computer Center
3202 E. Ashlan Ave.
Fresno, CA 93726
(209) 227-8479

On Line Computer Centers
5636 N. Blackstone
Fresno, CA 93710
(209) 432-4324

Glendale
Computerland Glendale
243 N. Brand Blvd.
Glendale, CA 91203
(213) 246-2453

Hundley Co.
933 Air Way
Glendale, CA 91201
(213) 240-3600

Goleta
Personal Electronics
5700 Calle Real
Goleta, CA 93117
(805) 967-5322

Hayward
Best Computer Store
1122 B St.
Hayward, CA 94541
(415) 537-2983

Huntington Beach
Gateway Computers
15201 Springdale
Huntington Beach, CA 92649
(714) 895-3931

Sun Computers
16241 Beach Blvd.
Huntington Beach, CA 92647
(714) 848-5574

Indio
Dean's Music City
82-704 Miles Ave.
Indio, CA 92201
(619) 347-5245

La Canada
Professional Computer Store
650 Foothill Blvd.
La Canada, CA 91011
(213) 248-2411

La Mesa
Computerland of San Diego East
7200 Parkway Dr.
La Mesa, CA 92041
(619) 464-5656

Laguna Hills
Computerland
Oakbrook Village, Ste. C-2
24241 Ave. De La Carolta
Laguna Hills, CA 92653
(714) 859-8912

Lawndale
The Computer Stop
16811 Hawthorne Blvd.
Lawndale, CA 90260
(213) 371-4010

Computerland, South Bay
16720 S. Hawthorne Blvd.
Lawndale, CA 90260
(213) 371-7144

Computique
16611 Hawthorne Blvd.
Lawndale, CA 90260
(213) 370-5795

Lomita
Sun Computers
1848 Pacific Coast Hwy.
Lomita, CA 90717
(213) 325-6200

Long Beach
A-VIDD Electronics Co.
2210 Bellflower Rd.
Long Beach, CA 90815
(213) 598-0444

Los Alamitos
Amis Desktop Computers
10512 Los Vaqueros Cir.
Los Alamitos, CA 90720
(714) 952-4122

Los Altos
Computerland of Los Altos
4546 El Camino Real
Los Altos, CA 94022
(415) 941-8154

Los Angeles
Byte Shop, Brentwood
11611 San Vicente Blvd.
Los Angeles, CA 90049
(213) 820-1524

Computer Showcase, Los Angeles
10517 W. Pico Blvd.
Los Angeles, CA 90064
(213) 474-6409

Compusystems Corp.
1039 W. 6th St.
Los Angeles, CA 90017
(213) 975-1220

Computerland of Los Angeles Downtown
650 S. Olive St.
Los Angeles, CA 90017
(213) 627-7154

Computerland of West Los Angeles
10600 Pico Blvd.
Los Angeles, CA 90064
(213) 559-3353

Computique
3285 Wilshire Blvd.
Los Angeles, CA 90010
(213) 385-7777

Computique
11986 Wilshire Blvd.
Los Angeles, CA 90025
(213) 820-5761

Computique, Downtown
435 W. 7th St.
Los Angeles, CA 90014
(213) 629-0121

Micropak Inc.
11862 La Grange Ave.
Los Angeles, CA 90025
(213) 472-0789

Los Gatos
Exectec
14103 Winchester Blvd.
Los Gatos, CA 95030
(408) 374-0170

Marysville
Computer Products Center
321 D St.
Marysville, CA 95901
(916) 743-6905

Mill Valley
Infomax
183 Lomita Dr.
Mill Valley, CA 94941
(415) 388-7775

Millbrae
Penninsula Computers
130 El Camino Real
Millbrae, CA 94030
(415) 692-7677

Modesto
Computer Ware
1031 15th St.
Modesto, CA 95354
(209) 578-9739

Monterey Park
Sun Computers Inc.
428 S. Atlantic Blvd., #102
Monterey Park, CA 91754
(213) 570-0901

Mountain View
Mission Computer Corp.
2065 W. El Camino Real, Ste. B
Mountain View, CA 94040
(415) 964-7063

Napa
Exectec
3202 Jefferson St.
Napa, CA 94558
(707) 253-0300

Newport Beach
A-VIDD Electronics Co.
4930 Campus Dr.
Newport Beach, CA 92660
(714) 851-1295

Computerland Newport Beach
4250 Scott Dr.
Newport Beach, CA 92660
(714) 975-0953

Newhall
Computer Store of Newhall
23422 Lyons Ave.
Newhall, CA 91321
(805) 254-0556

Computerland Santa Clarita Valley
23226 Lyons Ave.
Newhall, CA 91321
(805) 254-3121

Newark
Microland Computer Corp.
6050 Mowry Ave.
Newark, CA 94560
(415) 790-0410

Northridge
Rainbow Computing
9719 Reseda Blvd.
Northridge, CA 91324
(213) 349-5560

Oakland
Computer Store of Oakland
1320 Webster
Oakland, CA 94612
(415) 763-7900

Computerland of Oakland
366 Grand Ave.
Oakland, CA 94610
(415) 839-5230

Peninsula Office Supply, Oakland
200 Hegenberger Rd.
Oakland, CA 94621
(415) 638-5959

Quest Computer Store Inc.
2180 Franklin St.
Oakland, CA 94612
(415) 893-7381

Oceanside
Wabash Apple
2235 El Camino Real, Ste. A
Oceanside, CA 92054
(619) 721-0660

Orange
Apple of Orange Inc.
1904 Tustin Ave.
Orange, CA 92665
(714) 974-3082

Palo Alto
Macys California, Stanford
300 Stanford Shopping Ctr.
Palo Alto, CA 94309
(415) 326-3333

Mission Computer of Palo Alto
550 University Ave.
Palo Alto, CA 94301
(415) 326-9689

Peninsula Computer Co., Palo Alto Sales Office
175 Forest St.
Palo Alto, CA 94301
(415) 327-2775

Pasadena
Computerland
81 N. Lake St.
Pasadena, CA 91101
(213) 449-3205

Computique
260 S. Lake Ave.
Pasadena, CA 91101
(213) 795-3007

Di-No Computers
2499 E. Colorado Blvd.
Pasadena, CA 91107
(213) 795-4263

Paso Robles
The Computer Terminal
935 Riverside Ave., Ste. 10
Paso Robles, CA 93446
(805) 239-3136

Petaluma
Executron Systems Inc.
628 E. Washington St.
Petaluma, CA 94952
(707) 778-1242

Pleasanton
Home & Business Computer Center
1807-C Santa Rita Rd.
Pleasanton, CA 94566
(415) 846-3944

Rancho Cucamonga
CPU Business Systems
9155 Archibald, Ste. D
Rancho Cucamonga, CA 91730
(714) 989-8504

Redding
Computerland of Redding
1175 Hilltop Dr.
Redding, CA 96003
(916) 241-7922

Redondo Beach
Lamar Instruments
2107 Artesia Blvd.
Redondo Beach, CA 90278
(213) 374-1673

Redwood City
Penninsula Office Supply
1205 Veterans Blvd.
Redwood City, CA 94063
(415) 369-6761

Richmond
Vidiom
1220 Hilltop Mall Rd.
Richmond, CA 94806
(415) 223-1160

Ridgecrest
Compard
250 Balsam
Ridgecrest, CA 93555
(714) 375-8142

Riverside
Computer Kingdom
5225 Canyon Crest Dr., #30
Riverside, CA 92507
(714) 787-1142

Rocklin
Audio Video
3111 Sunset Blvd.
Rocklin, CA 95677
(916) 624-0601

Roseville
Capitol Computer Systems
212 Harding Blvd.
Roseville, CA 95678
(916) 786-8715

Sacramento
Capitol Computer Systems
1009 L St.
Sacramento, CA 95814
(916) 446-3101

Capitol Computer Systems
2800 Arden Way
Sacramento, CA 95825
(916) 483-7298

Computerland of Sacramento
1537 Howe Ave., #106
Sacramento, CA 95825
(916) 920-8981

Weinstocks
600 K St.
Sacramento, CA 95814
(916) 449-2447

Salinas
Peninsula Computer Center
208 Main St.
Salinas, CA 93901
(408) 424-2103

San Bernardino
Computerland of San Bernardino
289 E. Highland Ave.
San Bernardino, CA 92404
(714) 886-6838

San Diego
Byte Shop
8038 Clairemont Mesa Blvd.
San Diego, CA 92111
(619) 565-8008

Computer Age
4688 Convoy, #105
San Diego, CA 92111
(619) 565-4042

Computer Image
4603 Mission Bay Dr.
San Diego, CA 92109
(619) 270-3100

The Computer Merchant
5107 El Cajon Blvd.
San Diego, CA 92115
(619) 583-3963

Computer Store International
824 Camino Del Rio N. #329
San Diego, CA 92111
(619) 291-9531

Computerland San Diego
4237 Convoy
San Diego, CA 92111
(619) 560-9912

Orion Business Systems
11777 Bernardo Plaza Ct.
San Diego, CA 92128
(619) 485-8580

Wabash Computer Systems
4637 Convoy St., Ste. #101
San Diego, CA 92111
(619) 576-1604

San Francisco
The Computer Connection
214 California
San Francisco, CA 94111
(415) 781-0200

Computerland
117 Fremont St.
San Francisco, CA 94105
(415) 546-1592

Computerland, Market
2272 Market St.
San Francisco, CA 94114
(415) 864-8080

Computerland of San Francisco
1303 Van Ness Ave.
San Francisco, CA 94109
(415) 673-6640

Macys Computer Solutions
170 O'Farrell
San Francisco, CA 94120
(415) 393-3578

Quest Computers
710 Montgomery
San Francisco, CA 94111
(415) 982-3753

San Jose
Access Computers
5357 Prospect Rd.
San Jose, CA 95129
(408) 973-0111

Advanced Computer Products
542 W. Trimble
San Jose, CA 95131
(408) 946-7010

Computer Emporium
5821 Cottle Rd.
San Jose, CA 95123
(408) 227-5414

Computerland of Almaden
5035 Almaden Expwy.
San Jose, CA 95118
(408) 267-2182

Computerland San Jose
1077 Saratoga-Sunnyvale Rd.
San Jose, CA 95129
(408) 253-8080

Peninsula Office Supply
770 S. Bascom
San Jose, CA 95128
(408) 998-5415

Peninsula Office Supply
2043 Zanker Rd.
San Jose, CA 95131
(408) 288-7900

RAC Computers
1759 E. Capital Expwy.
San Jose, CA 95121
(408) 274-1915

San Luis Obispo
Coastal Computers
986 Monterey St.
San Luis Obispo, CA 93401
(805) 543-9339

San Mateo
Macys California, Hillsdale
115 Hillsdale Mall
San Mateo, CA 94403
(415) 341-3333

San Rafael
Computerland of Marin
835 4th St.
San Rafael, CA 94901
(415) 459-1767

San Ramon
Infomax
3124-D Crow Canyon Pl.
San Ramon, CA 94583
(415) 838-1027

Santa Ana
Advanced Computer Products
1310 E. Edinger, B
Santa Ana, CA 92705
(714) 558-8813

Computer City
3941 S. Bristol, D
Santa Ana, CA 92704
(714) 549-7749

Computique
3211 S. Harbor Blvd.
Santa Ana, CA 92704
(714) 549-7373

MTI-Microcomputer Technology
3304 W. MacArthur Blvd
Santa Ana, CA 92704
(714) 979-9923

Santa Barbara
Computer Plaza
3313-A State St.
Santa Barbara, CA 93105
(805) 687-9391

Computerland of Santa Barbara
3931 State St.
Santa Barbara, CA 93105
(805) 967-0413

Santa Clara
Affordable Computer Systems
3331 El Camino Real
Santa Clara, CA 95051
(408) 249-4221

Computerland of Santa Clara Valley
2037 El Camino Real
Santa Clara, CA 95051
(408) 246-4500

Santa Monica
The Computer Store
820 Broadway
Santa Monica, CA 90401
(213) 451-0713

Santa Maria
Computerland
223 S. Broadway
Santa Maria, CA 93454
(805) 928-1919

Santa Rosa
Computerland of Santa Rosa
611 5th St.
Santa Rosa, CA 95404
(707) 528-1775

Santa Rosa Computer Center
604 7th St.
Santa Rosa, CA 95404
(707) 528-6480

Simi Valley
Candid Computers
4390 Cochran St.
Simi Valley, CA 93063
(805) 522-3824

Stockton
Computers Etc.
7610 B-2 Pacific Ave.
Stockton, CA 95207
(209) 957-9500

Computerland of Stockton
4343 Pacific Ave., Ste. A-1
Stockton, CA 95207
(209) 473-1241

Stockton Computer
4555 N. Pershing, #27
Stockton, CA 95207
(209) 952-2028

Sunnyvale
Computer Plus Inc
1328 S. Mary Ave.
Sunnyvale, CA 94087
(408) 735-1199

Tarzana
Computique
18665 Ventura Blvd.
Tarzana, CA 91356
(213) 705-7507

Thousand Oaks
Computerland
171 E. Thousand Oaks Blvd., #104
Thousand Oaks, CA 91360
(805) 495-3554

Torrance
OMC Office Systems
20695 S. Western Ave., #124
Torrance, CA 90502
(213) 328-1760

Turlock
West Coast Computer Center
151 N. Thor
Turlock, CA 95380
(209) 667-1301

Tustin
Computerland of Tustin
104 W. Ist St.
Tustin, CA 92680
(714) 544-0542

Ukiah
Ukiah Computer Center
295 N. State St.
Ukiah, CA 95482
(707) 463-2556

Van Nuys
Compu-Plus
5848 Sepulveda Blvd.
Van Nuys, CA 91411
(213) 786-7411

Ventura
Computerland of Ventura
3875 Telegraph Rd., Ste. E
Ventura, CA 93003
(805) 656-7711

Visalia
West Coast Computer Center
3334 S. Mooney Blvd.
Visalia, CA 93277
(209) 739-1010

Walnut Creek
Computerland Walnut Creek
15 Ygnacio Valley Rd.
Walnut Creek, CA 94596
(415) 935-6502

Infomax
1300 Mt. Diablo Blvd.
Walnut Creek, CA 94596
(415) 935-5153

Peninsula Office Supply
2940 Main St.
Walnut Creek, CA 94596
(415) 945-0980

Westminster
Business Computer Center
14300 Beach Blvd
Westminster, CA 92683
(714) 892-0500

Computer Wave
6791 Westminster Ave.
Westminster, CA 92683
(714) 891-2584

West Covina
Computerland of West Covina
853 S. Glendora Ave
West Covina, CA 91790
(213) 960-6351

Whittier
Fyrst Byte Inc
10053 Whittwood Dr
Whittier, CA 90603
(213) 947-9411

Woodland Hills
Computerland, San Fernando Valley
20812 Ventura Blvd.
Woodland Hills, CA 91364
(213) 716-7714

Yucca Valley
Cameras & Computers Inc
57224 29 Palms Hwy
Yucca Valley, CA 92284
(714) 365-4005

Yucaipa
Yucaipa Computer Systems
32999 Yucaipa Blvd
Yucaipa, CA 92399
(714) 790-1237

COLORADO

Arvada
Computerland
8749 Wadsworth Blvd.
Arvada, CO 80005
(303) 420-1877

Aurora
Compu-Shop
3102 S. Parker Rd.
Aurora, CO 80014
(303) 420-1877

Micro Computer Center
2680 S. Havana, Ste. F
Aurora, CO 80014
(303) 751-0811

Boulder
Computer Works Inc.
3101 Walnut St.
Boulder, CO 80301
(303) 444-6550

The Computer Connection Ltd.
1600 38th St., Ste. 101
Boulder, CO 80301
(303) 449-8282

Colorado Springs
Academy Computers
320 N. Tejon
Colorado Springs, CO 80903
(303) 633-3600

Computerland
4543 Templeton Gap Rd.
Colorado Springs, CO 80918
(303) 574-4150

Denver
Computer Works Inc.
7400 E. Hampden
Denver, CO 80321
(303) 740-8650

Computerland
2422 S. Colorado Blvd.
Denver, CO 80222
(303) 759-4685

CW Electronics
800 Lincoln St.
Denver, CO 80203
(303) 832-1111

Idex Microsystems Inc.
999 18th St., Ste. 225
Denver, CO 80202
(303) 293-2299

Englewood
The Computer Connection Ltd
6818 S. Yosemite St.
Englewood, CO 80112
(303) 740-9360

Computerland
6801 Emporia St., Ste. 107
Englewood, CO 80112
(303) 850-7207

Evergreen
Computer Video Concepts (P)
Bear Creek Mall
Evergreen, CO 80252
(303) 674-2385

Computer Video Concepts (R)
4651 S. Colorado 73, Canyon Cir
Evergreen, CO 80439
(303) 674-8869

Fort Collins
Micro Computer Management
200 W. Prospect
Fort Collins, CO 80525
(303) 493-5700

Rocky Mountain Computer Works
2601 S. Lemay Ave.
Fort Collins, CO 80525
(303) 223-4000

Team Electronics #056
215 E. Foothills Pkwy.
Fort Collins, CO 80525
(303) 223-8326

Grand Junction
Computerland of West Colorado
644 Main St.
Grand Junction, CO 81501
(303) 245-2373

Team Electronic #067
Mesa Mall
Grand Junction, CO 81505
(303) 245-4455

Greeley
Micro Computer World Inc.
2331 27th St.
Greeley, CO 80631
(303) 330-1700

Team Electronics
2045 Greeley Mall
Greeley, CO 80631
(303) 356-3800

Lakewood
Computerland
85 K S. Union
Lakewood, CO 80228
(303) 988-0883

The Neighborhood Computer Store
13045 W. Alameda Pkwy.
Lakewood, CO 80228
(303) 988-9140

Littleton
Micro Concepts Inc.
5934 S. Kipling
Littleton, CO 80123
(303) 978-0537

Sterling
Computer West Inc.
1323 W. Main St.
Sterling, CO 80751
(303) 522-8886

Steamboat Springs
Pilot Office Supply
511 Lincoln
Steamboat Springs, CO 80477
(303) 879-3161

Westminster
Colorado Computer Systems
3005 W. 74th Ave.
Westminster, CO 80030
(303) 426-5880

HAWAII

Hilo
Computer Sales, Hawaii
200 Kanoelehua Ave., Waiakea Sq.
Hilo, HI 96720
(808) 935-9110

Honolulu
Computer Aiea
4510 Salt Lake Blvd.
Honolulu, HI 96818
(808) 487-0030

Computerland Hawaii
567 S. King, #132, Kawaiahao Plz.
Honolulu, HI 96813
(808) 521-8002

Memory Lane Computers
Pearl Harbor Naval Exchg.,
Johnson Cir.
Honolulu, HI 96818
(808) 422-2777

Memory Lane Computers Inc.
841 Bishop St., Ste. 153
Honolulu, HI 96813
(808) 526-3232

Microcomputer Systems
55 S. Kukui, C109
Honolulu, HI 96813
(808) 536-5288

Pineapple Computers Inc.
1833 Kalakaua Ave.
Honolulu, HI 96815
(808) 955-8858

Kailua-Kona
Computerworks, Hawaii
74-5616 Alapa St.
Kailua-Kona, HI 96740
(808) 329-8188

Kaneohe
Computerland of Hawaii, Kaneohe
46-208 Kahuhipa St., #101
Kaneohe, HI 96744
(808) 247-8541

IDAHO

Boise
Computerland
687 S. Capitol Blvd.
Boise, ID 83702
(208) 344-5545

Computerland, Boise
1006 W. Highlandview Dr.
Boise, ID 83702
(208) 342-4754

Northwest Computer Centre
6457 Fairview Ave.
Boise, ID 83704
(208) 375-6681

Idaho Falls
R & L Data Systems
522 Lomax St.
Idaho Falls, ID 83401
(208) 529-3785

Yost Office Systems
675 E. Anderson St.
Idaho Falls, ID 83401
(208) 523-3549

Lewiston
B & I Computer Systems
1824-B Main St.
Lewiston, ID 83501
(208) 746-5980

Team Electronics #160
Lewiston Shopping Ctr.
Lewiston, ID 83501
(208) 746-0086

Pocatello
Yost Office Systems
379 N. Yellowstone Hwy.
Pocatello, ID 83201
(208) 232-3931

Twin Falls
R & L Data Systems
108 W. Addison
Twin Falls, ID 83301
(208) 734-1357

MONTANA

Billings
The Computer Store
1216 16th W, Ste. 33
Billings, MT 59102
(406) 245-0092

Computerland
1827 Grand Ave.
Billings, MT 59101
(406) 259-0565

Team Electronics #168
Rimrock Mall
Billings, MT 59102
(406) 652-3070

Bozeman
Computer Lines Inc.
6 W. Main
Bozeman, MT 59715
(406) 586-7693

Great Falls
Emery Computers
9 3rd St. N
Great Falls, MT 59401
(406) 761-8954

Team Electronics #035
613 Central Ave.
Great Falls, MT 59401
(406) 453-3246

Kalispell
The Computer Place
36 2nd St. E
Kalispell, MT 59901
(406) 755-1323

Missoula
Business Machines Co. of Missoula
227 N. Higgins
Missoula, MT 59807
(406) 728-3741

Team Electronics #037
1801 Brooks St.
Missoula, MT 59801
(406) 549-4119

NEW MEXICO

Alamogordo
Computerworld
821 New York
Alamogordo, NM 88310
(505) 437-2273

Albuquerque
Computer Technology Associates
1704 Moon NE
Albuquerque, NM 87112
(505) 298-2140

Computerland, Albuquerque
2258 Wyoming NE
Albuquerque, NM 87112
(505) 294-2900

Microage of Albuquerque
5815 Menaul NE
Albuquerque, NM 87110
(505) 883-0955

Mnemonics Memory Systems
3100 Juan Tabo NE
Albuquerque, NM 87111
(505) 293-2983

Rocky Mountain Computer
2109 Wyoming NE
Albuquerque, NM 87110
(505) 292-2775

Clovis
Computer Shop
5500 Mabry Dr.
Clovis, NM 88101
(505) 762-3324

Farmington
Rocky Mountain Computer Co.
2014 San Juan Blvd., Ste. B
Farmington, NM 87401
(505) 326-1193

Hobbs
Electronic Module
601 N. Turner
Hobbs, NM 88240
(505) 397-3022

Las Cruces
Computer Technology Associates
1675 Lohman
Las Cruces, NM 88001
(505) 581-3500

Southwest Computer Center
121 Wyatt Dr., #7
Las Cruces, NM 88001
(505) 526-2842

Roswell
Southeastern Business Systems
103 E. 5th St.
Roswell, NM 88201
(505) 624-0732

Santa Fe
Computers Plus
1608 St. Michaels Dr.
Santa Fe, NM 87501

Computerland of Santa Fe
510 W. Cordova Rd., Coronado Ctr.
Santa Fe, NM 87501
(505) 988-8800

NEVADA

Carson City
A+ Computer
1213 S. Carson
Carson City, NV 89701
(702) 882-8686

Incline Village
Stereoscope, Microcomputer
930 Tahoe Blvd.
Incline Village, NV 89450
(702) 831-6499

Las Vegas
Century 23
4530 Meadows Ln., Ste. C1
Las Vegas, NV 89107
(702) 870-1534

Computerland
1370 E. Flamingo Rd., Ste. K
Las Vegas, NV 89109
(702) 369-2001

Home Computers
1775 E. Tropicana, #6
Las Vegas, NV 89109
(702) 798-1022

Reno
Byte Shop
4084 Kietzke Ln.
Reno, NV 89502
(702) 826-8080

Computerland of Reno
4914 S. Virginia St.
Reno, NV 89502
(702) 825-0378

OREGON

Albany
Team Electronics #154
1225 E. Pacific Blvd.
Albany, OR 97321
(503) 926-5900

Beaverton
Byte Shop, Beaverton
3482 S.W. Cedar Hills Blvd.
Beaverton, OR 97005
(503) 644-2686

Bend
Walmar Computers Inc.
1900 N.E. 1st, Ste. 202
Bend, OR 97701
(503) 382-2498

Canby
Team Electronics #151
1023 S.W. 1st
Canby, OR 97013
(503) 266-2539

Corvallis
Computer Store of Corvallis
2015 N.W. Circle Blvd.
Corvallis, OR 97330
(503) 754-0811

Eugene
Computer Solutions
175 Silver Ln.
Eugene, OR 97404
(503) 689-9677

The Computer Store, Eugene
35 W. 8th Ave.
Eugene, OR 97401
(503) 343-1434

Grants Pass
Team Electronics #162
530 N.E. E St.
Grants Pass, OR 97526
(503) 479-8723

Medford
Team Electronics #186
259 Barnett Rd., #3
Medford, OR 97501
(503) 773-9601

Pendleton
F&H Sound and Computers
338 S. Main St.
Pendleton, OR 97801
(503) 276-3772

Portland
Byte Shop, Portland
625 S.W. 10th Ave.
Portland, OR 97205
(503) 223-3496

Computerland, Multnomah County
327 S.W. Morrison St.
Portland, OR 97204
(503) 295-1928

The New Day Computing Co.
421 S.W. 5th
Portland, OR 97222
(503) 223-8033

Pegasus Computers
1517 N.E. 122nd Ave.
Portland, OR 97220
(503) 256-4713

Roseburg
Walmar Computers
1000 S.E. Stephens
Roseburg, OR 97470
(503) 673-6630

Salem
Computer Specialties Inc.
3390 Commercial St., SE
Salem, OR 97302
(503) 399-0534

Computerland of Salem
980 Lancaster Dr., NE
Salem, OR 97303
(503) 371-7070

Team Electronics
395 Liberty NE
Salem, OR 97301
(503) 371-7406

Team Electronics #129
2230 Fairground Rd., NE
Salem, OR 97303
(503) 364-3278

Tigard
Computerland
12020 S.W. Main St.
Tigard, OR 97223
(503) 620-6170

Tillamook
Bell's Office Products
1906 3rd St.
Tillamook, OR 97141
(503) 842-5566

UTAH

Cedar City
Personal Business Computers
197 W. 650 S
Cedar City, UT 84720
(801) 566-2648

Logan
Alpine Computing
851 N. Main
Logan, UT 84321
(801) 752-6432

Ogden
Computer Store
4421 Harrison Blvd.
Ogden, UT 84403
(801) 479-3131

Orem
Computer Technology Inc.
1455 S. State St.
Orem, UT 84057
(801) 224-1169

Provo
Allen's Computer Store
2230 N. University Pkwy., Ste. 11B
Provo, UT 84604
(801) 373-4443

Salt Lake City
Computer Solutions, Div. Inkley's
1984 S. State St.
Salt Lake City, UT 84115
(801) 467-5482

Computerland of Salt Lake City
161 E. 2nd S
Salt Lake City, UT 84111
(801) 364-4416

Management Systems Corp. (MSC)
200 E. South Temple
Salt Lake City, UT 84111
(801) 524-2000

Mnemonics Memory Systems
141 E. 200 S
Salt Lake City, UT 84111
(801) 266-7883

Personal Business Computers
1879 S. Main
Salt Lake City, UT 84115
(801) 486-4839

Vernal
Computer Store
46 S. 600 W
Vernal, UT 84078
(801) 789-9513

WASHINGTON

Bellevue
Computerland
14340 N.E. 20th
Bellevue, WA 98007
(206) 746-2070

Frederick and Nelson
Computer Depot
201 Bellevue Sq.
Bellevue, WA 98004
(206) 454-6311

Swan Computers Inc.
1034 116th Ave. NE
Bellevue, WA 98004
(206) 454-6272

Bellingham
Alpha Tech Computers
2300 James St., B1
Bellingham, WA 98225
(206) 647-2360

Computerland, Bellingham
1209 Cornwall Ave.
Bellingham, WA 98225
(206) 676-2522

Western Micro Computer Co.
4204 Guide Meridian
Bellingham, WA 98225
(206) 676-9558

Everett
Micro Computer Systems
1512 S. Everett Mall Way
Everett, WA 98904
(206) 355-4300

Federal Way
Computer Access Learning Center
32921 1st Ave. S
Federal Way, WA 98003
(206) 874-2540

Computerland
1500 S. 336th St.
Federal Way, WA 98003
(206) 838-9363

Kennewick
Alpha Computer Systems
1609 W. Kennewick Ave.
Kennewick, WA 99336
(509) 586-7603

Computerland, Kennewick
100 N. Morain St., Ste. 220
Kennewick, WA 99336
(509) 783-8174

Lacy
Stolz Computers Inc.
4106 Pacific Ave. SE
Lacy, WA 98503
(206) 459-9595

Longview
Team Electronics #161
1024 14th St.
Longview, WA 98632
(206) 425-3600

Lynnwood
Computerland of Lynnwood
18415 33rd Ave. W., Alderwood
Cor. Bldg.
Lynnwood, WA 98036
(206) 774-6993

Mount Vernon
Associated Business Machines
604 S. 3rd St.
Mount Vernon, WA 98273
(206) 336-9541

Tesarik Office Equipment
416 Gates St.
Mount Vernon, WA 98273
(206) 336-6111

Okanogan
ICM Computer Systems
716 1st Ave. S
Okanogan, WA 98840
(509) 422-6579

Olympia
Lakeside Computers
1200 Cooper Point Rd.
Olympia, WA 98502
(206) 786-0909

Port Angeles
Angeles Computer Inc.
540 E. 8th St.
Port Angeles, WA 98362
(206) 452-7881

Renton
Computerland of Renton
3200 N.E. Sunset Blvd., Sunset Plz.
Renton, WA 98055
(206) 271-8585

Richland
Alpha Computer Center, Richland
134 D George Washington Way
Richland, WA 98352
(509) 943-5608

Seattle
Byte Shop
2412 2nd Ave.
Seattle, WA 98121
(206) 622-7196

Computer Store of Seattle UW
1032 N.E. 65th
Seattle, WA 98115
(206) 522-0220

Computerland of Seattle
119 Yesler Way
Seattle, WA 98104
(206) 223-1075

Computershop Business Centers
11057 8th Ave. NE
Seattle, WA 98125
(206) 367-6800

Empire Electronics Inc.
616 W. 152nd
Seattle, WA 98166
(206) 244-5200

Silverdale
Computer Connection Inc.
3100 N.W. Bucklin Hill Rd.
Silverdale, WA 98383
(206) 692-7753

Spokane
The Computer Store of Spokane
W. 709 Riverside Ave.
Spokane, WA 99205
(509) 466-0446

Computerland of Spokane
E. 10623 Sprague Ave.
Spokane, WA 99206
(509) 924-4113

Electro-Mart
E. 3611 Sprague Ave.
Spokane, WA 99202
(509) 535-2451

Tacoma
Computerland
904 Broadway
Tacoma, WA 98402
(206) 383-4951

Start Computing
3802 S. Warner
Tacoma, WA 98409
(206) 475-5702

Tukwila
Computershop Business Centers
17125 Southcenter Pkwy.
Tukwila, WA 98188
(206) 575-1026

Vancouver
Computerland, Vancouver
6621 E. Mill Plain Blvd.
Vancouver, WA 98661
(206) 695-1540

Yakima
Cliff Miller's Cameras
2076 Yakima Mall
Yakima, WA 98901
(509) 575-8806

Cliff Miller's Cameras
22 N. 2nd St.
Yakima, WA 98901
(509) 248-1585

G. B. Services
1904 W. Nob Hill Rd.
Yakima, WA 98902
(509) 248-8622

WYOMING

Casper
Computerland of Casper
138 Kimball St.
Casper, WY 82601
(307) 234-2879

National Datatec
4055 Cy Ave.
Casper, WY 82604
(307) 265-8888

Team Electronics #126
207 S. Montana
Casper, WY 82601
(307) 235-6691

Cheyenne
Computerland of Cheyenne
3601 E. Lincoln Way
Cheyenne, WY 82001
(307) 634-9552

Gillette
Computerland of Gillette
801 E. 4th, Ste. 16
Gillette, WY 82716
(307) 682-6609

Rock Springs
Hi-Quality Office Machines and
Electronics
1405 Dewar Dr.
Rock Springs, WY 82901
(307) 382-2757

Central Region

ARKANSAS

Fayetteville
Alpha II Computer Concepts
1426 N. College
Fayetteville, AR 72701
(501) 442-8494

Fort Smith
Bits & Bytes Inc.
3000 Old Greewd Rd., Ste. C
Ft. Smith, AR 72903
(501) 646-5714

Little Rock
Computers Etc.
7624 Baseline Rd.
Little Rock, AR 72209
(501) 562-3200

Computerland
11121 Rodney Parham Rd.
Little Rock, AR 72212
(501) 224-4508

The Micro Computer Center
3502 S. University
Little Rock, AR 72204
(501) 565-3481

IOWA

Ames
Beacon Electronics
213 Lincoln Way
Ames, IA 50010
(515) 233-4807

Cyberia
2330 Lincoln Way
Ames, IA 50010
(515) 292-7634

Bettendorf
Memory Bank Inc
1721 Grant St.
Bettendorf, IA 52722
(319) 355-6401

Cedar Falls
Computerland, Cedar Falls
Blackhawk Village Ctr.
Cedar Falls, IA 50613
(319) 277-1700

Cedar Rapids
Midwest Office Systems
305 3rd Ave. SE
Cedar Rapids, IA 52401
(319) 366-7546

Team Electronics #036
4444 1st Ave. NE
Cedar Rapids, IA 52402
(319) 393-8956

Clinton
Clintons Computers Inc.
126 5th Ave. S
Clinton, IA 52732
(319) 242-2755

Council Bluffs
B & B Computer Systems Inc.
176 Midlands Mall
Council Bluffs, IA 51501
(712) 322-2600

Danbury
Rick's Computers
Hwy. 175
Danbury, IA 51019
(712) 883-2248

Davenport
Cinarco-Elliott A.V. & Computer
234 W. 3rd
Davenport, IA 52801
(319) 322-3163

Team Electronics #113
320 Kimberly Rd.
Davenport, IA 52806
(319) 386-2588

Des Moines
Computerland of Des Moines
7517 Douglas Ave.
Des Moines, IA 50322
(515) 270-8400

Dubuque
Computer Plus Inc.
3430 Dodge
Dubuque, IA 52001
(319) 556-6150

Team Electronics #007
2300 Kennedy Rd.
Dubuque, IA 52001
(319) 583-9195

Fairfield
Walkers Office Supplies
109-111 W. Broadway
Fairfield, IA 52556
(515) 472-2131

Fort Dodge
CKR Computer Systems Ltd.
809 Central Ave.
Fort Dodge, IA 50501
(515) 955-8300

Iowa City
North Bay Computerware
326 E. 2nd St.
Iowa City, IA 52240
(319) 337-2689

Team Electronics #093
The Mall, Room 120, Sp. 18
Iowa City, IA 52240
(319) 338-3681

Jefferson City
Lyon Company Inc.
116 E. State St.
Jefferson City, IA 50129
(515) 386-4276

Marion
Computerland of Marion
3271 Armar Dr.
Marion, IA 52302
(319) 373-1241

Marshalltown
Marshalltown Office Supply
20-24 Main St.
Marshalltown, IA 50158
(515) 752-4545

Mason City
Beacon Microcenter
122 W. State St.
Mason City, IA 50401
(515) 424-8205

Newton
Forbes Office Equipment Inc.
102 N. 2nd Ave. E
Newton, IA 50208
(515) 792-6168

Ottumwa
Meyer's Appliance
322 E. Main
Ottumwa, IA 52501
(515) 684-6547

Pella
B C S Computer Systems
514 Franklin St.
Pella, IA 50219
(515) 628-9295

Sioux City
Management Computer Systems
4415 Stone Ave.
Sioux City, IA 51106
(712) 274-1811

Storm Lake
Iowa Office Supply Inc.
731 Lake Ave.
Storm Lake, IA 50588
(712) 732-4801

Waterloo
Team Electronics #020
2750 University Ave.
Waterloo, IA 50701
(319) 235-6507

West Burlington
Midwest Office Systems Corp.
118 Wheeler
West Burlington, IA 52655
(319) 753-5448

West Des Moines
The Computer Emporium
801 73rd St.
West Des Moines, IA 50312
(515) 279-8861

ILLINOIS

Arlington Heights
Computerland
270 W. Rand Rd.
Arlington Heights, IL 60004
(312) 870-7500

Aurora
Farnsworth Computer Center
1891 N. Farnsworth
Aurora, IL 60505
(312) 851-3888

Belleville
Kappel's Computer Store
125 E. Main
Belleville, IL 62233
(618) 277-2354

Bethalto
Centaurus Computers
66 Plaza Dr.
Bethalto, IL 62010
(618) 377-2523

Bloomington
T Z Computers
1224 Towanda Ave., Ste. 21
Bloomington, IL 61701
(309) 829-6636

Buffalo Grove
Compushop
1363 W. Dundee Rd.
Buffalo Grove, IL 60090
(312) 577-0600

Carbondale
Illinois Computer Mart
Rte. 8, Sweet's Corner Plz.
Carbondale, IL 62901
(618) 529-2983

Centralia
Computer Management
Assistance Co.
420 S. Poplar
Centralia, IL 62801
(618) 533-0550

Champaign
Byte Shop, Champaign
313 N. Mattis
Champaign, IL 61820
(217) 352-2323

Computerland of Champaign
505 S. Mattis Ave.
Champaign, IL 61820
(217) 359-0895

Chicago
Chicago Computer Co.
222 N. Adams St.
Chicago, IL 60606
(312) 372-7360

Computer Room
638 N. Michigan Ave.
Chicago, IL 60611
(312) 337-7070

Computers Plus
5050 N. Cumberland Ave.
Chicago, IL 60656
(312) 452-0066

Erickson Communications
5456 N. Milwaukee
Chicago, IL 60630
(312) 631-5181

Systemsource
131 W. Madison St.
Chicago, IL 60603
(312) 726-7879

Collinsville
Centaurus Computers
838 S. Morrison Ave.
Collinsville, IL 62234
(618) 345-2523

Decatur
Main Street Computer Co.
215 N. Main
Decatur, IL 62523
(217) 429-5505

Team Electronics #079
Northgate Mall Shopping Ctr.
Decatur, IL 62526
(217) 877-2774

Deerfield
Video Etc.
465 Lake Cook Rd.
Deerfield, IL 60015
(312) 498-9669

Dekalb
Appletree Computer
1022 W. Lincoln Hwy.
Dekalb, IL 60115
(815) 758-8666

Downers Grove
Computerland
136 W. Ogden Ave.
Downers Grove, IL 60515
(312) 964-7762

Edwardsville
Computer Corner
332 Junction Dr.
Edwardsville, IL 62025
(618) 692-0622

Elmhurst
Computer Junction
543 S. York Rd.
Elmhurst, IL 60126
(312) 530-1125

Computerland of Elmhurst
192 N. York Rd.
Elmhurst, IL 60126
(312) 832-0075

Evanston
Computerland of Evanston
618 Davis St.
Evanston, IL 60201
(312) 328-3535

Nabih's Inc.
515 Davis St.
Evanston, IL 60201
(312) 869-6140

Galesburg
Midwest Information Systems
41 S. Cherry St.
Galesburg, IL 61401
(309) 342-7177

Team Electronics #141
1150 W. Carl Sandburg Dr.
Galesburg, IL 61401
(309) 344-1300

Geneva
Micro Computer Center
726 E. State
Geneva, IL 60134
(312) 232-1545

Glen Ellyn
Morsch-Netzel Enterprises
438 Roosevelt Rd.
Glen Ellyn, IL 60137
(312) 858-6692

Highland
Highland Computer Services
101 Forest Dr.
Highland, IL 62249
(618) 654-7465

Joliet
Computerland, Joliet
3121 W. Jefferson St.
Joliet, IL 60435
(815) 741-3303

Primus Equipment
101 Springfield Ave.
Joliet, IL 60435
(815) 741-4703

Kankakee
Ideal Computer Systems
101 S. Schuyler Ave.
Kankakee, IL 60901
(815) 935-8505

Lake Forest
Lakeshore Computers
1000 N. Western Ave.
Lake Forest, IL 60045
(312) 234-1002

Lincoln
Computers & Video Inc.
509 Pulaski St.
Lincoln, IL 62656
(217) 732-6717

Mattoon
Main Street Computer Store
1610 Broadway
Mattoon, IL 61938
(217) 234-4404

Midlothian
Compushop
14407 S. Cicero Ave.
Midlothian, IL 60445
(312) 396-1020

Moline
Team Electronics #117
4200 16th St.
S. Park Shopping Center
Moline, IL 61265
(309) 797-8261

Morton Grove
Compushop, Chicago
5920 W. Dempster St.
Morton Grove, IL 60053
(312) 967-0450

Mount Vernon
Computer Management Assistance Co.
104 N. 9th St.
Mount Vernon, IL 62864
(618) 242-4020

Mundelin
Computerland, Lake County
1500 S. Lake St.
Mundelin, IL 60060
(312) 949-1300

Naperville
Computerland, Naperville
1565 N. Naperville Rd.
Naperville, IL 60540
(312) 369-3511

Illini Microcomputers
630 E. Ogden Ave.
Naperville, IL 60540
(312) 420-8813

Niles
Computerland, Niles
9511 N. Milwaukee Ave.
Niles, IL 60648
(312) 967-1714

Normal
Appletree Computer
117 E. Beaufort
Normal, IL 61761
(309) 452-4215

Wallace Micro-Mart
106 Young
Normal, IL 61761
(309) 452-8665

Northbrook
Computerland, Northbrook
3069 Dundee Rd.
Northbrook, IL 60062
(312) 272-4703

Northbrook Computers
4113 Dundee Rd., Sanders Crt.
Northbrook, IL 60062
(312) 480-9190

Oakbrook Terrace
Oak Brook Computer Centre
17 W. 426 22nd St.
Oakbrook Terrace, IL 60181
(312) 941-9005

Oak Lawn
Computerland, Oak Lawn
10935 S. Cicero Ave.
Oak Lawn, IL 60453
(312) 422-8080

Oak Park
Computerland, Oak Park
965 Lake St.
Oak Park, IL 60301
(312) 383-1606

Bies Systems Inc.
7037 W. North Ave.
Oak Park, IL 60302
(312) 386-3323

Orland Park
Microage, Orland Park
8752 W. 159th St.
Orland Park, IL 60462
(312) 349-8080

Video Etc. Inc.
9107 151 St.
Orland Park, IL 60462
(312) 460-8980

Paris
Parkway Computers
Hwy. #1 N
Paris, IL 61944
(217) 465-4733

Peoria
Computerland, Peoria
4507 N. Sterling
Peoria, IL 61614
(309) 688-6252

Wallace Micro-Mart Inc.
2619 N. University
Peoria, IL 61604
(309) 685-7876

Posen
Computers Etc.
2515 W. 147th Pl.
Posen, IL 60469
(312) 388-2000

Quincy
Sauer Computer Systems
2080 Broadway
Quincy, IL 62301
(217) 228-1135

Rockford
Alpine Computer Center
2418 S. Alpine Rd.
Rockford, IL 61108
(815) 229-0200

Computer Store of Rockford
3515 Auburn St.
Rockford, IL 61103
(815) 962-7580

Rolling Meadows
Compushop
1219 E. Golf Rd.
Rolling Meadows, IL 60008
(312) 593-1800

Saint Charles
Computerland, Saint Charles
2075 Prairie St., Ste. 111
Saint Charles, IL 60174
(312) 377-7200

Schaumburg
Computerland, Schaumburg
1097 E. Golf Rd.
Schaumburg, IL 60195
(312) 843-7740

Data Domain
1612 E. Algonquin Rd.
Schaumburg, IL 60195
(312) 397-8700

Digital Den
E302 Woodfield Mall
Schaumburg, IL 60195
(312) 884-8520

Omega Computers Inc.
1123 Salem Plz.
Schaumburg, IL 60194
(312) 884-0430

Springfield
Main Street Computer Co.
323 E. Monroe
Springfield, IL 62701
(217) 753-0426

Micropower Computer System
923 S. 5th St.
Springfield, IL 62703
(217) 544-4108

Team Electronics #185
2501 W. Wabash/White Oaks Mall
Springfield, IL 62704
(217) 793-3505

Team Electronics #071
2716 S. MacArthur Blvd.
Springfield, IL 62704
(217) 525-8637

Sterling
Computer Base
1112 E. 4th
Sterling, IL 61081
(815) 626-4115

Taylorville
Main Street Computer #4
123 W. Main Cross
Taylorville, IL 62568
(217) 824-4984

Villa Park
Farnsworth Computer Center
383 E. North Ave.
Villa Park, IL 60181
(312) 833-7100

INDIANA

Anderson
Computerland, Anderson
1003 Meridian
Anderson, IN 46016
(317) 649-1122

Bloomington
Data Domain
221 W. Dodds St.
Bloomington, IN 47401
(812) 334-3607

Columbus
Micro Computer Systems Inc.
2626 Eastbrook Plz.
Columbus, IN 47201
(812) 522-2280

Elkhart
General Microcomuter #3
1500 Oslo Rd., Ste. 4C
Elkhart, IN 46514
(219) 277-4972

Evansville
Better Business Computers #2
1500 N. Greenriver Rd.
Evansville, IN 47715
(812) 479-3500

Computerland of Evansville
725 S. Green River Rd.
Evansville, IN 47715
(812) 473-3303

Fort Wayne
Computerland, Fort Wayne
5450 N. Coldwater Rd.
Fort Wayne, IN 46825
(219) 483-8107

The Data Base
3426 N. Anthony Blvd.
Fort Wayne, IN 46805
(219) 484-3164

Graham Electronics #2
3433 E. Washington
Fort Wayne, IN 46803
(800) 348-4619

Greenwood
Computer Experience
1265 N. Madison Ave.
Greenwood, IN 46142
(317) 887-0797

Indianapolis
Computer Management Systems
6801 Lake Plaza Dr., Ste. B-206
Indianapolis, IN 46220
(317) 842-1830

Computerland, Indianapolis
6040 E. 82nd St.
Indianapolis, IN 46250
(317) 849-8811

Graham Electronics #1
133 S. Pennsylvania
Indianapolis, IN 46204
(317) 634-8202

Graham Electronics #4
9449 Aronson Dr.
Indianapolis, IN 46240
(317) 844-1255

Microage Computer Store
8615 Allisonville Rd.
Indianapolis, IN 46250
(317) 849-5161

Kokomo
Beckley Office Equipment
112 S. Main St.
Kokomo, IN 46901
(317) 457-5571

Lafayette
Computer Age
2632-C N. 9th St.
Lafayette, IN 47905
(317) 423-5436

Digital Technology
10 N. 3rd St.
Lafayette, IN 47901
(317) 423-2548

Marion
The Computer Store
935 N. Baldwin Ave.
Marion, IN 46952
(317) 662-9994

Merrillville
Computerland
19 W. 80th Pl.
Merrillville, IN 46410
(219) 769-8020

Michigan City
General Microcomputer #4
9th & York St.
Michigan City, IN 46637
(219) 874-5231

Mishawaka
Computerland, South Bend
719 W. McKinley Ave.
Mishawaka, IN 46544
(219) 256-5688

Muncie
The Data Base
3032 N. Granville Rd.
Muncie, IN 47303
(317) 284-8900

Munster
Meade Electric
921-C Ridge Rd.
Munster, IN 46321
(219) 836-5350

New Albany
Paris Office Systems
410 Pearl St.
New Albany, IN 47150
(812) 944-7827

Richmond
Rosa's Inc.
20 S. 11 St.
Richmond, IN 47374
(317) 962-5543

South Bend
Computer Room #3
6341 University Commons
South Bend, IN 46635
(219) 277-1515

General Micro Computer #2
121 S. Niles Ave.
South Bend, IN 46601
(219) 233-7118

General Microcomputer #1
52303 Emmons Rd., #11
South Bend, IN 46637
(219) 277-4972

Terre Haute
Hoosier Electronics #1
9 Meadows Shopping Ctr.
Terre Haute, IN 47803
(812) 238-1456

Hoosier Electronics #2
J-7 Honey Creek Sq.
Terre Haute, IN 47803
(812) 232-8508

Vincennes
Better Business Computer #1
509 N. 7th St.
Vincennes, IN 47591
(812) 882-1512

West Lafayette
Computerland, West Lafayette
1020 Sagamore Pkwy.
West Lafayette, IN 47906
(317) 463-3546

KANSAS

Coffeyville
Computer Associates
124 W. 8th
Coffeyville, KS 67337
(316) 251-1880

Colby
Western Office Supply Inc.
1675 W. 4th
Colby, KS 67701
(913) 462-3923

Dighton
Central Computer Systems
110 E. Pearl
Dighton, KS 67839
(316) 386-2101

Dodge City
Dodge City Office Equipment
606 2nd
Dodge City, KS 67801
(316) 227-3106

El Dorado
Laforge's Inc
121-123 N. Main
El Dorado, KS 67042
(316) 321-3100

Emporia
Mr. Computer
1424 Industrial Rd.
Emporia, KS 66801
(316) 342-4893

Garden City
Team Electronics #046
215 W. Kansas Ave.
Garden City, KS 67846
(316) 276-2911

Hays
Northwestern Business Systems
800 Main St.
Hays, KS 67601
(913) 625-7323

Hutchinson
Computerland of Hutchinson
2522 N. Main St.
Hutchinson, KS 67501
(316) 662-6832

Lawrence
Computerland of Lawrence
1420 W. 23rd St.
Lawrence, KS 66044
(913) 841-4611

Liberal
Tradewind Systems
#52 Village Plz.
Liberal, KS 67901
(316) 624-8111

Mankato
Bronco Computer Systems
116 W. Jefferson
Mankato, KS 66956
(913) 378-3117

Mission
Computertrend
6500 Martway
Mission, KS 66202
(913) 722-2030

Overland
On Line Computer Centers
8935 S. Metcalf Ave.
Overland, KS 66212
(913) 341-6651

Overland Park
Beatty Electronics
7105 W. 105th St.
Overland Park, KS 66212
(913) 341-3500

Computerland
10049 Sante Fe Dr.
Overland Park, KS 66212
(913) 492-8882

Personal Computer Center
3819 95th St.
Overland Park, KS 66206
(913) 649-5942

Phillipsburg
Modern Business Machines
329 F St.
Phillipsburg, KS 67661
(913) 543-5206

Pittsburg
Barney and Associates
113 W. 5th
Pittsburg, KS 66762
(316) 231-1970

Prairie Village
Amerisource
5344 W. 95th St.
Prairie Village, KS 66207
(913) 642-1015

Salina
Microage Computer Store
1805 S. 9th
Salina, KS 67401
(913) 823-7596

Shawnee
Custom Computers
12107 Johnson Dr.
Shawnee, KS 66216
(913) 631-7600

Topeka
Computerland
911 A S W 37th
Topeka, KS 66611
(913) 267-6530

Wichita
Computerland, Wichita
6100 E. Central
Wichita, KS 67208
(316) 684-3870

Hitec Computer Center
1038 W. Pawnee
Wichita, KS 67213
(316) 262-0315

Hitec Computer Center
1210 Rock Rd.
Wichita, KS 67207
(316) 685-1131

Wilbur E. Walker Co. Inc.
522 S. Market
Wichita, KS 67201
(316) 267-2231

KENTUCKY

Bowling Green
Bowman-Kelley Inc.
400 E. Main St.
Bowling Green, KY 42101
(502) 782-3600

Frankfort
Computer Connections
#4 Fountain Pl.
Frankfort, KY 40601
(502) 875-1351

Hopkinsville
Randolph-Hale
1825 E. 9th St.
Hopkinsville, KY 42240
(502) 885-5357

Lexington
CBM Inc
198 Moore Dr.
Lexington, KY 40503
(606) 276-1519

The Computer Place
2551 Regency Rd.
Lexington, KY 40503
(606) 276-3592

Computerland, Lexington
380 W. Main St.
Lexington, KY 40507
(606) 231-9333

Lexington Computer Store
2909 Richmond Rd.
Lexington, KY 40509
(606) 268-1431

Louisville
Computer Emporium #1
620 S. 5th St.
Louisville, KY 40202
(502) 589-9482

Computer Emporium #2
203 Whittington Pkwy.
Louisville, KY 40222
(502) 432-7120

The Computer Store of Louisville
4820 U.S. Hwy. 42
Louisville, KY 40222
(502) 423-0690

Computerland, Louisville
10414 Shelbyville Rd.
Louisville, KY 40223
(502) 245-8288

Paducah
Cagle Business Systems
311 Kentucky Ave.
Paducah, KY 42001
(502) 442-9331

Carron Copy Systems
410 Broadway
Paducah, KY 42001
(502) 444-7737

LOUISIANA

Alexandria
Delta Microcomputer
3844 Independence Dr.
Alexandria, LA 71301
(318) 442-0217

Baton Rouge
The Computer Place
5500 Florida Blvd.
Baton Rouge, LA 70821
(504) 926-4630

Gretna
Calculator & Computer Center
2003 Daniels Rd.
Gretna, LA 70053
(504) 368-6586

Houma
The Computer People
111 Front Dr.
Houma, LA 70360
(504) 851-5253

Lafayette
The Computer People Inc.
3541 Ambassador Caffery Pkwy.
Lafayette, LA 70503
(504) 385-5175

The Computer Place
133 James Comeaux
Lafayette, LA 70508
(318) 232-4097

Lake Charles
Computer Concepts
3218 Ryan St.
Lake Charles, LA 70601
(318) 439-5627

Metairie
Calculator & Computer Center
3303 Severn Ave.
Metairie, LA 70002
(504) 888-2590

Computer Shoppe Inc.
3828 Veteran's Blvd.
Metairie, LA 70002
(504) 454-6600

Computerland, New Orleans
3517-19th St.
Metairie, LA 70002
(504) 456-1438

Micro Computer Store
4539 Interstate 10
Metairie, LA 70002
(504) 885-5883

Minden
The Office Machine Store
112 Pearl St.
Minden, LA 71058
(318) 377-2222

Monroe
Monroe Office Equipment
1001 N. 11th St.
Monroe, LA 71201
(318) 388-4600

Morgan City
The Computer People Inc.
Inglewood Mall
Morgan City, LA 70380
(504) 385-5175

New Orleans
Calculator & Computer Center
208 O'Keefe Ave.
New Orleans, LA 70112
(504) 522-7332

Calculator & Computer Center
9954 Lake Forest Blvd., Ste. 8
New Orleans, LA 70127
(504) 241-2207

Computerland of Louisiana
312 St. Charles Ave.
New Orleans, LA 70130
(504) 522-2255

The Computer Connection
5630 Jefferson Hwy.
New Orleans, LA 70123
(504) 733-5434

Shreveport
Computer Services of Shreveport
4436-A Youree Dr.
Shreveport, LA 71105
(318) 865-7189

Micro Business Systems
3823 Gilbert
Shreveport, LA 71104
(318) 226-8848

MICHIGAN

Ann Arbor
Complete Computer Center
413 E. Huron
Ann Arbor, MI 48104
(313) 994-6344

Computer Mart #6
2711 Plymouth Rd.
Ann Arbor, MI 48105
(313) 665-4453

Computerland of Ann Arbor
3410 Washtenaw Ave.
Ann Arbor, MI 48104
(313) 973-7075

Battle Creek
Krum's Photographic Inc.
35 E. Michigan Mall
Battle Creek, MI 49017
(616) 962-9525

Birmingham
Simtec Inc.
4114 W. Maple Rd.
Birmingham, MI 48010
(313) 855-3990

East Lansing
Computer Mart of Lansing
1331 E. Grand River
East Lansing, MI 48923
(517) 351-1777

Escanaba
Team Electronics #074
Delta Plaza Shopping Ctr.
Escanaba, MI 49829
(906) 786-3911

Farmington Hills
Computer Connection #1
38437 Grand River
Farmington Hills, MI 48018
(313) 477-4470

Flint
Computer Mart Inc.
915 S. Dort Hwy., #E
Flint, MI 48503
(313) 234-0161

Computer Mart (West)
3292 S. Linden Rd.
Flint, MI 48507
(313) 234-7791

Garden City
Retail Computer Center #1
28251 Ford Rd.
Garden City, MI 48135
(313) 422-2570

Gaylord
The Computer Haus
118 E. Main St.
Gaylord, MI 49735
(517) 732-6544

Grand Blanc
Computer Contact Inc.
3017 E. Hill Rd.
Grand Blanc, MI 48439
(313) 694-3740

Grand Rapids
Computer Room #2
3740 28th St.
Grand Rapids, MI 49508
(616) 949-2802

Hancock
Sayen Business Equipment
208 Quincy St.
Hancock, MI 49930
(906) 482-0612

Holt
Computer Connection #2
2495 N. Cedar St., Suite 7A
Holt, MI 48842
(517) 694-4594

Kalamazoo
Computer Mart of Kalamazoo
7428 S. Westnedge
Kalamazoo, MI 49002
(616) 329-1000

Computer Room #1
455 W. Michigan
Kalamazoo, MI 49007
(616) 343-4634

Kentwood
Computerland, Grand Rapids
2927 28th St. SE
Kentwood, MI 49508
(616) 942-2931

Livonia
Computer Horizons
37099 Six Mile Rd.
Livonia, MI 48152
(313) 464-6502

Muskegon
Advanced Management Systems
2084 Henry St.
Muskegon, MI 49441
(616) 759-8700

Saginaw
Computer Mart #2
3580 Bay Rd.
Saginaw, MI 48603
(517) 790-1360

Saint Clair Shore
Computerland, Grosse Pointe
22000 Greater Mack Ave.
Saintt Clair Shore, MI 48080
(313) 772-6540

Sanford
CAI Instruments Inc.
152 E. Saginaw
Sanford, MI 48657
(517) 835-6145

Southfield
Computerland
29673 Northwestern Hwy.
Southfield, MI 48034
(313) 356-8111

Spectrum Business Systems
26618 Southfield Rd.
Southfield, MI 48076
(313) 552-9092

Spring Lake
Mill Point Computer
206 W. Savidge St.
Spring Lake, MI 49456
(616) 846-4410

Traverse City
Team Electronics #159
1180 S. Airport Rd. W
Traverse City, MI 49456
(616) 946-8326

Troy
Computer Mart #5
1824 W. Maple Rd.
Troy, MI 48084
(313) 649-0910

Rainbow Computers Inc.
819 E. Big Beaver
Troy, MI 48084
(313) 528-3535

Warren
Lyceum Inc.
28657 Hoover Rd.
Warren, MI 48093
(313) 574-2444

West Bloomfield
Retail Computer Center #2
4381 Orchard Lake Rd.
West Bloomfield, MI 48033
(313) 855-4220

MINNESOTA

Albert Lea
Info Pro
110 W. Pearl St.
Albert Lea, MN 56007
(507) 377-1810

Alexandria
Alexandria Data Systems
411 Broadway
Alexandria, MN 56308
(612) 762-2138

Blaine
Digital Den
349 Northtown Dr.
Blaine, MN 55434
(612) 786-6044

Blooming Prairie
Let 3 Computer Service & Systems
Hwy. 218S & Main St.
Blooming Prairie, MN 55917
(507) 583-2494

Bloomington
Computerland, Bloomington
8070 Morgan Circle Dr.
Bloomington, MN 55431
(612) 884-1474

Brooklyn Center
Zim Computers
5717 Xerxes Ave. N
Brooklyn Center, MN 55429
(612) 560-0336

Burnsville
Audio King, Burnsville
14250 Burnhaven Rd.
Burnsville, MN 55337
(612) 435-8933

Computer Professionals
14322 Burnhaven Dr.
Burnsville, MN 55337
(612) 435-8060

Dayton's Computer Center
14204 Burnhaven Dr.
Burnsville, MN 55337
(612) 435-8811

Digital Den
2138 Burnsville Ctr.
Burnsville, MN 55337
(612) 435-5445

Team Electronics #148
1032 Burnsville Ctr.
Burnsville, MN 55337
(612) 435-7128

Duluth
Compudata Corp.
104 W. Superior
Duluth, MN 55802
(218) 722-6319

Digital Den
1600 E2 Miller Trunk Hwy.
Duluth, MN 55811
(218) 253-4434

Team Electronics #165
504 E. 4th St.
Duluth, MN 55805
(218) 727-4900

Edina
Audio King
7101 France Ave. S
Edina, MN 55435
(612) 920-0505

Dayton's Computer Center
100 Southdale Ctr.
Edina, MN 55435
(612) 375-4200

Team Electronics #095
204 Southdale Ctr.
Edina, MN 55435
(612) 920-4817

Fairbault
United Managers Inc.
303 N.E. 1st Ave., Depot Sq.
Faribault, MN 55021
(507) 332-2241

Fergus Falls
Team Electronics #163
Westridge Mall
Fergus Falls, MN 56357
(218) 739-4443

Hibbing
Team Electronics #125
Mesabi Mall
Hibbing, MN 55746
(218) 263-8200

Hopkins
Computerland, Hopkins
11319 Hwy. 7
Hopkins, MN 55343
(612) 933-8822

Hutchinson
Hutch Computer Industries
1115 W. Hwy. 7
Hutchinson, MN 55350
(612) 587-2940

Mankato
Team Electronics #023
Madison E
Mankato, MN 56001
(507) 387-7937

Tri-State Digital Products
510 Long St.
Mankato, MN 56001
(507) 625-8181

Maplewood
Digital Den
1009 Maplewood Mall
Maplewood, MN 55109
(612) 770-3424

Personal Business Systems
2225 White Bear Ave.
Maplewood, MN 55109
(612) 770-6160

Team #184
2162-63 Maplewood Mall
Maplewood, MN 55109
(612) 770-5513

Minneapolis
Audio King, Brookdale
5515 Xerxes Ave. N
Minneapolis, MN 55430
(612) 566-2360

Audio King, Ridgedale
1808 S. Plymouth Rd.
Minneapolis, MN 55400
(612) 546-4040

Blumberg Photo Sound Co.
525 N. Washington Ave.
Minneapolis, MN 55401
(612) 333-1271

Computer Professional
1200 Nicollet Mall
Minneapolis, MN 55403
(612) 341-8297

Computerland of Minneapolis
121 S. 8th St., Ste. 240
Minneapolis, MN 55402
(612) 333-3151

Dayton's Computer Center
700 Nicollett Mall
Minneapolis, MN 55402
(612) 375-2008

Micro Age Computer Store
83 S. 10th St.
Minneapolis, MN 55403
(612) 338-1777

Personal Business Systems
4306 Upton Ave. S
Minneapolis, MN 55410
(612) 929-4120

Personal Business Systems
801 N. 77½ St.
Minneapolis, MN 55423
(612) 866-3441

Team Electronics #001
2640 Hennepin Ave. S
Minneapolis, MN 55408
(612) 377-9840

Team Electronics #005
6413 Lyndale Ave. S
Minneapolis, MN 55423
(612) 869-3288

Team Electronics #050
1311 4th St. SE
Minneapolis, MN 55414
(612) 378-1185

Minnetonka
Audio King
1808 S. Plymouth Rd.
Minnetonka, MN 55343
(612) 546-4040

Dayton's Computer Center
12411 Wayzata Blvd.
Minnetonka, MN 55343
(612) 375-5500

Team Electronics #120
12503 Wayzata Blvd.,
Ridgedale Mall
Minnetonka, MN 55343
(612) 544-7412

Owatonna
Team Electronics #133
Cedar Mall
Owatonna, MN 55060
(507) 451-7248

Rochester
Computerland, Rochester
12th St. and S. Broadway
Rochester, MN 55901
(507) 281-0968

Digital Den
610 Apache Mall
Rochester, MN 55901
(507) 281-2261

Digital Designs
1137 N.W. 6th St.
Rochester, MN 55901
(507) 282-3222

Roseville
Audio King
1723 W. County Rd., B2
Roseville, MN 55113
(612) 636-8525

Dayton's Computer Center
900 Rosedale Ctr.
Roseville, MN 55113
(612) 375-6200

Digital Den
408 Rosedale Ctr.
Roseville, MN 55112
(612) 636-2631

Team Electronics #066
350 Rosedale Ctr.
Roseville, MN 55113
(612) 636-5147

Saint Anthony
Team Electronics #013
Apache Plaza, Silver Lake Rd.
Saint Anthony, MN 55421
(612) 789-4368

Saint Cloud
Digital Den
813 Saint Germain St.
Saint Cloud, MN 56301
(612) 253-4434

Team Electronics #004
3341 W. Saint Germain
Saint Cloud, MN 56301
(612) 251-1335

Team Electronics #131
Crossroads Shopping Ctr.
Saint Cloud, MN 56301
(612) 253-8326

Saint Paul
Computer Castle Inc.
240 Town Sq.
Saint Paul, MN 55101
(612) 227-3526

Digital Den
58 Snelling S
Saint Paul, MN 55105
(612) 699-8442

Personal Business Systems
2067 Ford Pkwy.
Saint Paul, MN 55116
(612) 698-1211

Schaak Electronics
1415 Mendota Heights Rd.
Saint Paul, MN 55120
(612) 454-6830

Team Electronics #002
455 Rice St.
Saint Paul, MN 55103
(612) 227-7223

Virginia
Team Electronics #121
Thunderbird Mall
Virginia, MN 55792
(218) 741-5919

Waite Park
Computerland, Saint Cloud
240 2nd Ave. S
Waite Park, MN 56387
(612) 259-0590

West Saint Paul
Team Electronics #058
1733 S. Robert St.
West Saint Paul, MN 55118
(612) 451-1765

Willmar
Tri-State Digital
1409 S. Hwy. 71
Willmar, MN 56201
(612) 235-8181

Woodbury
Computers of Woodbury
1750-21 Weir Dr.
Woodbury, MN 55125
(612) 739-2767

Worthington
Tri-State Digital Products
1024 Oxford St.
Worthington, MN 56187
(507) 372-7362

MISSOURI

Ballwin
Computer Country West
720 Manchester
Ballwin, MO 63011
(314) 394-3300

Cape Girardeau
Carron Copy Systems
351 N. Kingshighway
Cape Girardeau, MO 63701
(314) 334-1252

Computer Mart of Cape Girardeau
209 S. Plaza Way
Cape Girardeau, MO 63701
(618) 529-2983

Columbia
Century Next Computers
1001 E. Walnut, #202
Columbia, MO 65201
(314) 442-6502

Team Electronics #089
301 Stadium Blvd
Columbia, MO 65201
(314) 445-4496

Creve Couer
Forsythe Computers
11445 Olive St. Rd
Creve Couer, MO 63141
(314) 567-0450

Ellisville
Lifestyle Computers
1370 Clarkson
Ellisville, MO 63011
(314) 227-5577

Florissant
Computer Country North
235 Dunn Rd.
Florissant, MO 63031
(314) 921-4434

Gladstone
Computerland
7638 N. Oak Traffic Way
Gladstone, MO 64118
(816) 436-3737

Independence
Computerland
1214 A S. Noland Rd
Independence, MO 64055
(816) 461-6502

Jefferson City
Century Next Computers
2120 Missouri Blvd.
Jefferson City, MO 65201
(314) 636-6502

Joplin
Computer Solutions
1102 Range Line Rd
Joplin, MO 64801
(417) 623-2089

The Computer Patch
3316 E. 32nd St.
Joplin, MO 64802
(417) 782-0285

Kansas City
Computerage Business Systems
1104 Grand Ave.
Kansas City, MO 64106
(816) 421-3040

Computers Asp
7115 N.W. Barry Rd.
Kansas City, MO 64152
(816) 741-8013

The Computer Workshop Inc
4027 N. Oak Trafficway
Kansas City, MO 64116
(816) 452-3690

Kirksville
Computer Core
1902 S. Baltimore
Kirksville, MO 63501
(816) 627-1255

Kaleidoscope
1316 N. Baltimore
Kirksville, MO 63501
(816) 665-1953

Liberty
The Bottom Line
316 S. 291 Hwy.
Liberty, MO 64068
(816) 792-1500

Manchester
Computerland of Manchester
Carafiol Plz., 1327 Manchester Rd
Manchester, MO 63011
(314) 227-8088

United Computer Store
1056 E. Manchester
Manchester, MO 63011

Maryland Heights
Skarbek Computers Computerland
11990 Dorsett Rd
Maryland Heights, MO 63043
(314) 567-3291

Saint Charles
United Computer Store
2220 1st Capital
Saint Charles, MO 63301
(314) 928-1266

Saint Joseph
Brown Business Systems
920 S. 6th St.
Saint Joseph, MO 64501
(816) 279-0855

Computerland
2304 N. Belt
Saint Joseph, MO 64506
(816) 364-4498

Saint Louis
Computer Country
1015 Locust
Saint Louis, MO 63101
(314) 231-1101

Computer Country South
4479 Lemay Ferry Rd.
Saint Louis, MO 63129
(314) 487-2033

Computer Station
11610 Page Service Dr
Saint Louis, MO 63141
(314) 432-7019

Famous Barr Computer Center
Northwest Plaza Ctr
Saint Louis, MO 63074
(314) 291-6404

Famous Barr Computer Center
601 Olive (Downtown)
Saint Louis, MO 63101
(314) 241-5469

Forsythe Computers
521 Olive St
Saint Louis, MO 63101
(314) 721-4300

Forsyth Computers
7748 Forsyth Ave.
Saint Louis, MO 63105
(314) 721-4300

Lifestyle Computers
621 Westport Plz.
Saint Louis, MO 63141
(314) 469-0555

Springfield
Computer Mart
1904B E. Meadowmere
Springfield, MO 65804
(417) 862-6500

Computerland, Springfield
3640 S. Campbell
Springfield, MO 65807
(417) 887-2222

Database Systems
1550 E. Battlefield
Springfield, MO 65804
(417) 883-5665

NORTH DAKOTA

Bismarck
Computerland, Bismarck
531 Airport Rd.
Bismarck, ND 58501
(701) 224-0008

Team Electronics #030
2304 E. Broadway
Bismarck, ND 58501
(701) 223-4546

Dickinson
Team Electronics #167
194 & North Dakota State 22,
Sp. 127
Dickinson, ND 58601
(701) 225-4464

Fargo
Computer 1 Inc
Village West
Fargo, ND 58102
(701) 282-8820

Computerland, Fargo
3217 13th Ave. S
Fargo, ND 58103
(701) 237-3069

Digital Den
I-29 & 13th S
Fargo, ND 58103
(701) 282-9171

Grand Forks
Computerland, Grand Forks
2500-B S. Columbia Rd.
Grand Forks, ND 58201
(701) 746-0491

Computerland, Minot
1015 N. 39th St.
Grand Forks, ND 58201
(701) 746-0491

Team Electronics #026
1503 11th Ave., Box 1277
Grand Forks, ND 58201
(701) 746-4474

Team Electronics #157
Columbia Mall, Sp. 109
Grand Forks, ND 58201
(701) 775-5512

Minot
Team Electronics #032
910 24th Ave. SW
Minto, ND 58701
(701) 852-3281

Wahpeton
Malme Equipment Inc
Hwy. 13 W
Wahpeton, ND 58075
(701) 642-9229

Williston
Team Electronics #146
109 Main St.
Williston, ND 58801
(701) 572-7631

NEBRASKA

Bellevue
The Computer Works
119 W. Mission
Bellevue, NE 68005
(402) 291-7809

Columbus
Advanced Computer Systems
2402 13th St.
Columbus, NE 68601
(402) 564-9393

Fremont
Fremont Office Equipment Co
648 N. Broad
Fremont, NE 68025
(402) 721-6436

Gering
Panhandle Technical Systems
1925 10th
Gering, NE 69341
(308) 436-3436

Grand Island
Eakes Office Products Center
617 W. 3rd St
Grand Island, NE 68802
(308) 382-8026

Team Electronics #137
148 Conestoga Mall, Hwy. 281 & 13
Grand Island, NE 68801
(308) 381-0559

Kearney
Computer Hardware
2415 Central Ave.
Kearney, NE 68847
(308) 234-9335

Lincoln
Alcorn Video Center
301 S. 70th St., Ste. 110,
Med. Plz. Bldg.
Lincoln, NE 68510
(402) 489-0309

Computer Systems Inc.
940 N. 27th St.
Lincoln, NE 68503
(402) 474-2800

Computerland of Lincoln
701 N. 48th
Lincoln, NE 68504
(402) 467-5277

Electronics Center
1840 O St.
Lincoln, NE 68508
(402) 476-7331

Microtech
200 N. 66th St.
Lincoln, NE 68505
(402) 467-5521

Team Electronics #027
127 S. 19th St.
Lincoln, NE 68510
(402) 435-2959

Norfolk
Appletree Software-Consulting
1511 Riverside Blvd.
Norfolk, NE 68701
(402) 379-4020

Team Electronics #140
Sunset Plaza SC
Norfolk, NE 68701
(402) 379-1161

North Platte
Computer Center
105 E. 2nd St.
North Platte, NE 69101
(308) 532-6971

Team Electronics #149
1000 S. Dewey, The Mall
North Platte, NE 69101
(308) 534-4645

Omaha
Abacus Inc
1209 Harney, Lower Level
Omaha, NE 68102
(402) 345-6020

Computer Solutions Inc.
4217 S. 84th St.
Omaha, NE 68127
(402) 339-7441

Computers West
7423 Pacific St.
Omaha, NE 68114
(402) 393-2100

Computerland
11031 Elm St.
Omaha, NE 68144
(402) 391-6716

Database Systems
2754 S. 129th St.
Omaha, NE 68144
(402) 330-3600

Wayne
The Computer Farm
110 Main St.
Wayne, NE 68787
(402) 375-4331

OHIO

Akron
The Basic Computer Shop
2671 W. Market St.
Akron, OH 44313
(216) 867-0808

Archbold
Wyse Book and Office Supply
1409 S. Defiance St.
Archbold, OH 43502
(419) 445-0565

Ashtabula
Mace Electronics #5
4900 N. Ridge W
Ashtabula, OH 44004
(216) 969-1313

Athens
Vere Smith Audio Visuals Inc
16 W. Union St.
Athens, OH 45701
(614) 593-7708

Beaver Creek
Micro Computer Center #2
3255 E. Patterson
Beaver Creek, OH 45430
(513) 429-9355

Beechwood
Basic Computer Shop #3
28700 Chagrin Blvd.
Beechwood, OH 44122
(216) 831-3882

Belpre
Best Office Machines Inc.
616 Main St.
Belpre, OH 45714
(614) 423-9566

Bowling Green
Sound Associates #1
248 S. Main St.
Bowling Green, OH 43402
(419) 352-3595

Canton
Basic Computer Shop #2
2802 Whipple Rd. NW
Canton, OH 44708
(216) 478-0056

Computer World Inc.
4218 Hills and Dales Rd. NW
Canton, OH 44708
(216) 478-0033

Computerland of Akron, Canton
4106 Belden Village St. NW
Canton, OH 44718
(216) 493-7786

Chillicothe
The Chillicothe Computer Store
25 E. 2nd St.
Chillicothe, OH 45601
(614) 774-6565

Cincinnati
Abacus Computer Store
227 E. 6th St.
Cincinnati, OH 45202
(513) 421-5900

Cincinnati Computer Store
1711 Princeton Pike
Cincinnati, OH 45246
(513) 671-6440

Computerland of Cincinnati
19 W. 7th St.
Cincinnati, OH 45202
(513) 381-3844

Computerland, NE Cincinnati
9873 Montgomery Rd.
Cincinnati, OH 45242
(513) 984-3721

Future Now Shops
7336 Kenwood Rd.
Cincinnati, OH 45236
(513) 791-4700

Graham Electronics #3
239 Northland Blvd.
Cincinnati, OH 45246
(513) 772-1661

Micro Systems Management
520 Ohio Pike
Cincinnati, OH 45230
(513) 528-4202

Cleveland
Basic Computer Shop #4
238 Euclid Ave.
Cleveland, OH 44114
(216) 241-0030

J R Holcomb's
3000 Quigley Rd.
Cleveland, OH 44113
(216) 621-6580

North Coast Computers
646 Dover Center Rd.
Cleveland, OH 44140
(216) 835-4345

Columbus
Ads Office Machines
642 W. Broad St.
Columbus, OH 43215
(614) 224-8823

Computerland
6429 Busch Rd
Columbus, OH 43229
(614) 888-2215

Microage Computer Store
2591 Hamilton Rd.
Columbus, OH 43227
(614) 868-1550

On Line Computer Center
5866 Columbus Sq.
Columbus, OH 43229
(614) 895-7747

The Micro Center
1555 W. Lane Ave.
Columbus, OH 43221
(614) 486-5381

Dayton
Computer Solutions
1 E. Stewart St.
Dayton, OH 45409
(513) 223-2348

Micro Computer Center #1
7900 Paragon Rd.
Dayton, OH 45459
(513) 435-9355

Roth Office Equipment Co.
108 N. Jefferson St.
Dayton, OH 45402
(513) 228-6175

Findlay
Sound Associates #3
400 S. Main
Findlay, OH 45840
(419) 424-1191

Hudson
Hudson Computer Systems
205 Main St.
Hudson, OH 44236
(216) 653-9010

Lima
Computer Connection
1206 W. Robb Ave.
Lima, OH 45805
(419) 222-6464

Massillon
M H Martin Co.
1531 Amherst Rd. NE
Massillon, OH 44646
(216) 832-7467

Mayfield Heights
Computerland, Cleveland East
1288 Som Center Rd.
Mayfield Heights, OH 44124
(216) 461-1200

Mentor
Cleveland Computer Co.
7673 Mentor Ave.
Mentor, OH 44060
(216) 946-1722

Miamisburg
Computerland of Dayton
#17 Prestige Plz.
Miamisburg, OH 45342
(513) 439-1000

North Olmsted
Computerland, Cleveland West
4579 Great Northern Blvd.
North Olmsted, OH 44070
(216) 777-1433

Toledo
Abacus II Microcomputers #1
1417 Bernath Pkwy.
Toledo, OH 43615
(419) 865-1009

Abacus II Microcomputers #2
4751 Monroe St.
Toledo, OH 43623
(419) 471-0082

Sound Associates
5206 Monroe
Toledo, OH 43623
(419) 885-3547

Warren
Computerland, Warren
2000 North Rd. SE
Warren, OH 44484
(216) 544-4191

Wooster
Computer Directions
243 E. Liberty St.
Wooster, OH 44691
(216) 264-5383

Youngstown
Computerland, Youngstown
813 Boardman Poland Rd.
Youngstown, OH 44512
(216) 758-7569

OKLAHOMA

Ada
Computer Utility Corp.
808 E. Main St.
Ada, OK 74820
(405) 332-4858

Enid
Contemporary Sounds Inc.
432 S. Van Buren
Enid, OK 73701
(405) 233-3883

Guymon
Advanced Computer Systems
519 N. Main
Guymon, OK 73942
(405) 338-3253

Lawton
Computer Technology of Lawton
1720 A Cache Rd.
Lawton, OK 73501
(405) 353-2554

Midwest City
Computer Solutions Inc.
105 W. Atkinson Plz
Midwest City, OK 73110
(405) 733-9556

Muskogee
Data/Cap Inc. Computers
212 N. Main
Muskogee, OK 74401
(918) 683-1185

Norman
Computers/Associates
2301 W. Main
Norman, OK 73069
(405) 360-6818

Pro-Am Photo & Microcomp
513 W. Gray
Norman, OK 73069
(405) 364-5992

Oklahoma City
Computer Connection
7720 N. Robinson, #B7
Oklahoma City, OK 73116
(405) 842-4480

Computer Connections
12314 N. May Ave.
Oklahoma City, OK 73120
(405) 755-9220

Computer Connections
1029 S.E. 66th St.
Oklahoma City, OK 73149
(405) 632-3020

Computer Solutions
6214 N.W. Expwy.
Oklahoma City, OK 73132
(405) 722-7133

Computerland
10621 N. May Ave.
Oklahoma City, OK 73120
(405) 755-5200

Computerland of Crossroads
7812 S. Western
Oklahoma City, OK 73139
(405) 634-4300

EMS Computer Corp.
1145 S.W. 74th St., Bldg D
Oklahoma City, OK 73139
(405) 636-1504

Team Electronics #115
7000 Crossroads, Crossroads Mall
Oklahoma City, OK 73149
(405) 634-3357

Team Electronics #181
2501 W. Memorial, Quail Spg. Mall
Oklahoma City, OK 73120
(405) 751-2035

Stillwater
Stillwater Typewriter Co.
125 S. Main St.
Stillwater, OK 74074
(405) 372-3246

Tulsa
Computer Connections
8125 A E. 51st St.
Tulsa, OK 74145
(918) 663-6342

Computerland
8191 S. Harvard, WC II Ctr.
Tulsa, OK 74136
(918) 481-0332

The Computer Store
5153 S. Peoria
Tulsa, OK 74105
(918) 747-9333

SOUTH DAKOTA

Aberdeen
Computer Specialists
519 S. Lincoln
Aberdeen, SD 57401
(605) 229-3788

Pierre
Team Electronics #130
402 W. Sioux Ave.
Pierre, SD 57501
(605) 224-1881

Rapid City
Computer Systems Design
2139 Jackson Blvd.
Rapid City, SD 57701
(605) 341-3662

Computerland, Rapid City
738 Saint Joe St.
Rapid City, SD 57701
(605) 348-5384

Team Electronics #040
1101 Omaha St.
Rapid City, SD 57701
(605) 343-8363

Sioux Falls
Computer Terminal Ltd. 10
1800 S. Minnesota Ave.
Sioux Falls, SD 57105
(605) 335-6464

Computerland, Sioux Falls
3518 S. Western Ave.
Sioux Falls, SD 57105
(605) 338-5263

Team Electronics #132
4001 W. 41st, Sioux Emp. Mall
Sioux Falls, SD 57106
(605) 339-3730

Watertown
Team Electronics #019
223 9th Ave. SE
Watertown, SD 57201
(605) 886-4725

Yankton
Team Electronics #164
Yankton Mall
Yankton, SD 57078
(605) 665-8402

TEXAS

Abilene
Computer Shop, Abilene
917 N. Judge Ely Blvd.
Abilene, TX 79601
(915) 673-6708

Computerland of Abilene
4522 Buffalo Gap Rd.
Abilene, TX 79605
(915) 695-6110

Amarillo
Computerland of Amarillo
2300 Bell St., Interstate Vlg. SC
Amarillo, TX 79106
(806) 353-7482

Computer Corner
1800 S. Georgia
Amarillo, TX 79109
(806) 355-5618

Arlington
Compushop
2813 Galleria, Great SW Plaza
Arlington, TX 76011
(214) 265-5571

Computer Port
2142 N. Collins
Arlington, TX 76011
(817) 469-1502

Austin
ABC Computers & Telephone
9012 Research, Ste. 11
Austin, TX 78758
(512) 458-4236

The Computer Center
3736 Bee Cave Rd.
Austin, TX 78746
(512) 327-5864

Computers 'N' Things
2825 Hancock Dr.
Austin, TX 78731
(512) 453-5970

Computerland
3300 Anderson Ln., Shoal Crk.
Austin, TX 78757
(512) 452-5701

Computerland of Austin
3201 Bee Cave Rd
Austin, TX 78746
(512) 327-7044

Computer Resources
4211 S. Lamar & Ben White
Austin, TX 78746
(512) 443-3624

Beaumont
Computer Concepts
4699 Calder Ave.
Beaumont, TX 77707
(713) 892-3992

Bellaire
Compushop
5315 Bissonnet
Bellaire, TX 77401
(713) 661-2005

Brownsville
Computerland of Brownsville
3302 Boca Chica (United Plaza)
Brownsville, TX 78521
(512) 541-9261

College Station
Young Electronics Services
1804 Brothers Blvd.
College Station, TX 77840
(713) 693-8080

Corsicana
Central Texas Business Machines
421 N. Beaton St.
Corsicana, TX 75110
(214) 872-7435

Corpus Christi
Microcomputer Shoppe
5301 Everhart
Corpus Christi, TX 78411
(512) 855-4516

Dallas
Compco
5519 Arapaho Rd., #108
Dallas, TX 75248
(214) 386-6578

Compushop
8211 Preston Rd., Preston Ctr.
Dallas, TX 75225
(214) 739-0822

Compushop, Addison
5290 Beltline Rd., #116
Dallas, TX 75240
(214) 934-3107

Compushop, Keystone
13929 N. Central Expwy.
Dallas, TX 75243
(214) 234-3412

Computer Center of Dallas
2629 Stemmons Frwy., #215
Dallas, TX 75207
(214) 638-4477

Computer Wares Inc.
12300 Inwood Rd., #106
Dallas, TX 75234
(214) 960-0800

Computerland, Dallas
8061 Walnut Hill Ln., #8
Dallas, TX 75234
(214) 363-2223

Simtec Management Corp.
12801 Midway Rd., Ste. 509
Dallas, TX 75234
(214) 484-3311

Denton
Applied Data Systems
1811 N. Elm
Denton, TX 79410
(817) 566-6205

El Paso
Computer Technology Associates
118 Castelland
El Paso, TX 79912
(915) 581-3500

Computer Technology Associates
9530 Viscount
El Paso, TX 79925
(915) 593-6655

Computerland
3737 N. Mesa St.
El Paso, TX 79902
(915) 533-6060

Mnemonics Memory Systems
1400 Airway Blvd., Airway Jct Mall
El Paso, TX 79925

Fort Worth
Compushop
6353 Camp Bowie Blvd.
Fort Worth, TX 76116
(817) 738-4442

Computer Pro
516 Main St.
Fort Worth, TX 76102
(817) 334-0098

Computer Wares Inc.
4760 S. Hulen, Hulen Park
Shopping Ctr.
Fort Worth, TX 76132
(817) 346-0446

Computerland of Fort Worth
6275 Old Gransbury Rd.
Fort Worth, TX 76133
(817) 292-7114

Hardin Electronics
5635 E. Rosedale
Fort Worth, TX 76112
(817) 429-9761

Professional Micro Computing
6867A Greenoaks Rd.
Fort Worth, TX 76116
(817) 654-3360

Houston
Compushop
5900 N. Freeway
Houston, TX 77076
(713) 699-5301

Compushop
211-A FM 1960 W. Cypress
Houston, TX 77090
(713) 893-2060

Compushop, Downtown
815 Milam
Houston, TX 77002
(713) 227-1523

Computer Center of Houston
2200 Southwest Frwy., Ste. 120
Houston, TX 77098
(713) 527-8008

Computer City
12704 North Frwy.
Houston, TX 77060
(713) 821-2702

Computer Craft
1958 W. Gray
Houston, TX 77019
(713) 522-3130

Computer Craft
2709 Chimney Rock
Houston, TX 77056
(713) 840-9762

Computer Craft
10615 Katy Frwy.
Houston, TX 77056
(713) 827-1744

Computer Craft
3233 Fondren
Houston, TX 77063
(713) 977-0664

Computer Craft Inc.
5050 FM 1960 W. Ste. 104A
Houston, TX 77069
(713) 583-2032

Computer Galleries
2493 S. Braeswood
Houston, TX 77030
(713) 661-0055

Computer Gallaries
11538 Northwest Frwy.
Houston, TX 77092
(713) 956-0900

Computer Wares
12839 Gulf Freeway
Houston, TX 77034
(713) 947-9633

Computerland
17647 El Camino, Comino V
Houston, TX 77058
(713) 488-8153

Computerland Houston Southwest
6100 Westheimer, #138A
Houston, TX 77057
(713) 977-0909

Simtec Management Corp.
1990 E. Post Oak Blvd.
Houston, TX 77056
(713) 850-9797

Humble
Supertec of Texas Inc.
262 FM 1960 E. By Pass
Humble, TX 77338
(713) 446-9770

Hurst
Microage Computer Store
1220 Melbourne Dr.
Hurst, TX 76053
(817) 284-3413

Irving
Computer Wares
2209 Story Rd.
Irving, TX 75062
(214) 258-0080

Compushop, Las Colinas
Las Colinas Tower IV
125 Carpenter Frwy.
Irving, TX 75062
(214) 556-2166

Irving Computers
1770 W. Irving Blvd., Unit 7
Irving, TX 75061
(214) 254-6850

Laredo
Computerland, Laredo
4500 San Bernardo Ave., Ste. 11A
Laredo, TX 78041
(512) 724-1551

Southwest Computer Center
5603 N. 35, Ste. 1
Laredo, TX 78041
(512) 724-1133

Longview
Computerland of Longview
1905 N.W. Loop 281
Longview, TX 75601
(214) 297-0145

Lubbock
Agriplex Computer
3206 A 34th St.
Lubbock, TX 79410
(806) 797-4495

Computerland, Lubbock
6223 Slide Rd.
Lubbock, TX 79414
(806) 792-3835

Lufkin
Computerland of Lufkin
3047 Angelina Mall
Lufkin, TX 75901
(713) 632-8100

Marshall
Marshall Business Equipment
607 Pinecrest Dr. W
Marshall, TX 75670
(214) 938-8371

McAllen
Computerland of McAllen
3000 N. 10th St.
McAllen, TX 78501
(512) 686-3743

On-Line Computer Center
2300D N. 10th St.
McAllen, TX 78501
(512) 631-2321

Midland
Computers Ltd.
409 Andrews Hwy.
Midland, TX 79703
(915) 686-9819

Computer Technology Associates
San Miguel Sq., Ste. 115
Midland, TX 79703
(915) 699-5046

Nacogdoches
Spinet Music
2819 North St.
Nacogdoches, TX 75961
(713) 564-8311

North Richardson Hills
Compu Shop
8214 Bedford-Euless Rd.
N. Richardson Hills, TX 76118
(817) 498-8106

Odessa
Computer Patch
3952 E. 42nd St.
Odessa, TX 79762
(915) 563-3506

Paris
Mech-Tronics Business Systems
1838 Bonham
Paris, TX 75460
(214) 785-7991

Plano
Compushop, Plano
3100 Independence Pkwy., #316
Plano, TX 75075
(214) 867-4595

Computer Video Systems
1301 Custer Pkwy., #348
Plano, TX 75075
(214) 423-3654

Computer Wares
1915 N. Central Expwy
Plano, TX 75074
(214) 422-5584

Richardson
Computerland
1535 Promenade Ctr.
Richardson, TX 75080
(214) 235-1285

San Antonio
Com-Sol Computer Solutions
4801 Fredericksburg Rd
San Antonio, TX 78229
(512) 341-8851

Computerland of San Antonio
8373 Perrin Beitel
San Antonio, TX 78218
(512) 654-4886

Computershop
6901 Blanco
San Antonio, TX 78216
(512) 340-1979

Computershop
5011 Walzem
San Antonio, TX 78218
(512) 657-7034

Expensive Toys For Big Boys
2060 Music Ct., N. Star Mall
San Antonio, TX 78216
(512) 340-5600

Sherman
Computerland, Texoma Vis Comm
2816 Hwy. 75 N
Sherman, TX 75090
(214) 893-0111

Systems Services
4520 Hwy. 75 N
Sherman, TX 75020
(214) 892-4652

Texarkana
FNS Computer Services Inc.
1911 N. Robison Rd.
Texarkana, TX 75501
(214) 832-5666

Tyler
Computer Electronics
1607 S. Holt
Tyler, TX 75701
(214) 597-3700

Computerland, Texoma Vis Comm
225 E. Amherst
Tyler, TX 75701
(214) 581-7000

Victoria
Computer Command
708 E. Goodwin
Victoria, TX 77902
(512) 573-4305

Waco
Computer Solutions Inc.
711 Lake Air Dr.
Waco, TX 76710
(817) 776-7655

Computerland of Waco
522 Meadowlake Ctr.
Waco, TX 76710
(817) 776-6700

Waco Communications Inc.
401 W. Loop 340
Waco, TX 76710
(817) 772-8550

Webster
Compushop
229 Nasa Rd., 1 E
Webster, TX 77598
(713) 332-9628

Wichita Falls
The Computer Store
3725 Call Field Rd.
Wichita Falls, TX 76308
(817) 691-4552

WISCONSIN

Appleton
Computer World
3015 W. Wisconsin
Appleton, WI 54911
(414) 733-9547

Eau Claire
Computerland, Eau Claire
1404 S. Hastings
Eau Claire, WI 54701
(715) 835-8082

Digital Den
2844 London Square Dr.
Eau Claire, WI 54701
(715) 832-9698

Team Electronics #006
2321 E. Clairemont Pkwy.
Eau Claire, WI 54701
(715) 834-1288

Greenbay
Computer World
1800 W. Mason St.
Greenbay, WI 54303
(414) 499-9983

Greendale
Schaak Electronics
5300 S. 76th St., Southridge
Greendale, WI 53129
(414) 421-6555

Team Electronics #068
5300 S. 76th St., Southridge Mall
Greendale, WI 53129
(414) 421-4300

Greenfield
Byte Shop of Milwaukee
4840 S. 76th
Greenfield, WI 53220
(414) 281-7004

Digital Den
4625 S. 27th St.
Greenfield, WI 53221
(414) 281-3123

Janesville
Computerland of Janesville
2517 Milton Ave.
Janesville, WI 53545
(608) 752-1177

Team Electronics #088
2619 Milton Ave.
Janesville, WI 53545
(608) 756-3150

La Crosse
Century Computer
2935 East Ave. S
La Crosse, WI 54601
(608) 788-2350

Computerland, La Crosse
1711 George St.
La Crosse, WI 54601
(608) 781-2090

Team Electronics #171
3800 Hwy. 16, #154,
Valley View Mall
La Crosse, WI 54601
(608) 781-8500

Madison
American Computer Systems
702 E. Washington Ave.
Madison, WI 53703
(608) 257-1348

Blue Lakes Computing
3240 University Ave.
Madison, WI 53705
(608) 233-6502

Computerland, Madison
6625 Odana Rd.
Madison, WI 53719
(608) 833-8900

Computerland, Madison East
3205 E. Washington Ave.
Madison, WI 53704
(608) 241-2100

Team Electronics #008
3365 E. Washington Ave.
Madison, WI 53704
(608) 244-1339

Marinette
Team Electronics #029
2510 Roosevelt Rd
Marinette, WI 54143
(715) 732-4421

Mequon
Computerland of Ozaukee County
6111 W. Mequon Rd.
Mequon, WI 53092
(414) 242-9490

Milwaukee
Compco
7110 W. Fond Du Lac
Milwaukee, WI 53218
(414) 438-0610

Computerland, Milwaukee
10111 W. Capitol Dr.
Milwaukee, WI 53222
(414) 466-8990

Computers Plus Inc.
2749 S. 108th St.
Milwaukee, WI 53227
(414) 321-1770

Computers Unltd. of Wisconsin
780 N. Jefferson
Milwaukee, WI 53202
(414) 355-5206

Computers Unltd. of Wisconsin
8360 W. Browndeer Rd.
Milwaukee, WI 53223
(414) 355-5206

Micro Age Computer Store
2675 N. Mayfair Rd.
Milwaukee, WI 53226
(414) 251-1100

North Shore Computers Inc.
5261 Port Washington Rd
Milwaukee, WI 53217
(414) 963-9700

Team Electronics #097
7700 W. Browndeer Dr.
Milwaukee, WI 53223
(414) 354-4880

Neenah
Computerlab
885 S. Greenbay Rd.
Neenah, WI 54956
(414) 725-3020

Office Technology Inc.
1314 S. Commercial St.
Neenah, WI 54956
(414) 725-5551

Oshkosh
Computerland, Fox River Valley
1526 S. Keoller, Menard Plz.
Oshkosh, WI 54901
(414) 233-1808

Racine
Colortron Computer
2111 Lathrop
Racine, WI 53405
(414) 637-2003

Rhinelander
Team Electronics #128
Hwy. 8 E, Sunrise Plz.
Rhinelander, WI 54501
(715) 369-3900

Sheboygan
Ross Business Equipment
1406 N. 25th St.
Sheboygan, WI 53081
(414) 452-2271

Superior
Digital Den
46 Mariner Mall
Superior, WI 54880
(715) 392-7175

Waukesha
Blue Lakes Computing
20300 W. Bluemound Rd
Waukesha, WI 53218
(414) 785-1788

Wausau
Computerland Wausau
2424 Stewart Ave.
Wausau, WI 54401
(715) 842-0348

Team Electronics #018
2207 Grand Ave.
Wausau, WI 54401
(715) 842-3364

WEST VIRGINIA

Barboursville
Solutions Inc
3542 Rte. 60 E
Barboursville, WV 25504
(304) 736-0762

Charleston
The Computer Store
Municipal Park Bldg., Ste. 5
Charleston, WV 25301
(304) 345-1360

Clarksburg
Computerland, Clarksburg
403 W. Main St.
Clarksburg, WV 26301
(304) 624-6409

Media Pack
822 W. Pike St.
Clarksburg, WV 26301
(304) 622-2211

Huntington
Computer Store
541 9th St.
Huntington, WV 25701
(304) 529-6426

Morgantown
The Computer Corner
22 Beechurst Ave.
Morgantown, WV 26505
(304) 292-9700

Sound and Electronic Specialists
Rte. #12
Morgantown, WV 26505
(304) 594-1855

Parkersburg
Bobeir TV Rental Inc.
3701 Murdock Ave.
Parkersburg, WV 26101
(304) 485-7802

Computerland Parkersburg
3415 Murdock Ave.
Parkersburg, WV 26101
(304) 485-6823

South Charleston
Computerland of Charleston
224 7th Ave.
South Charleston, WV 25303
(304) 744-7962

Eastern Region

ALABAMA

Andalusia
The Computer House
921 River Falls St.
Andalusia, AL 36420
(205) 222-7753

Anniston
Computerland
301 S. Quintard Ave.
Anniston, AL 36201
(205) 237-5600

Kemp's Officenter
1201 Noble St.
Anniston, AL 36202
(205) 236-6396

Birmingham
AC3 Computing Products Center
2029 2nd Ave. N
Birmingham, AL 35203
(205) 251-4330

AC3 Computing Products Center
1580 Montgomery Hwy.
Birmingham, AL 35226
(205) 822-2533

Computerland
215 W. Valley Ave.
Birmingham, AL 35209
(205) 942-8085

Village Computers
1720 28th Ave. S
Birmingham, AL 35209
(205) 870-8943

Demopolis
Collins Communications
1009 W. Jackson St.
Demopolis, AL 36732
(205) 289-0439

Dothan
Computer Solutions Inc.
301 S. Oates St.
Dothan, AL 36301
(205) 793-0049

Florence
Anderson Computers
1826 Darby Dr.
Florence, AL 35630
(205) 767-4660

Gadsden
Computers Plus
226 S. 5th St.
Gadsden, AL 35902
(205) 543-2703

Huntsville
Anderson Computers
3156 University Dr. NW
Huntsville, AL 35805
(205) 539-3444

Mobile
Computerland
3696 Airport Blvd.
Mobile, AL 36608
(205) 342-2540

Olensky Bros Inc.
3763 Airport Rd.
Mobile, AL 36608
(205) 344-7448

Montgomery
AC3 Computing Products Center
Union Sq., 157 Eastern By-Pass
Mongtomery, AL 36117
(205) 271-2211

Computer Solutions
1744 Narrow Lane Rd.
Montgomery, AL 36106
(205) 263-9269

Northport
Syscon
1609 Bridge Ave.
Northport, AL 35476
(205) 339-8241

Tuscaloosa
AC3 Computing Products Center
2600 McFarland Blvd. E
Tuscaloosa, AL 35405
(205) 758-2825

CONNECTICUT

Bethel
Technology Systems
208 Greenwood Ave.
Bethel, CT 06801
(203) 748-6856

Branford
Advanced Office Systems
116 N. Main St.
Branford, CT 06405
(203) 481-5349

Danbury
Computerland of Danbury
76 West St.
Danbury, CT 06810
(203) 748-2300

East Hartford
Computerease Inc. II
84 Connecticut Blvd.
East Hartford, CT 06108
(203) 278-1080

Fairfield
Computerland
1700 Post Rd., Heritage Sq.
Fairfield, CT 06430
(203) 255-9252

The Micro Computer Store
1905 Black Rock Tnpk.
Fairfield, CT 06430
(203) 335-3694

Farmington
Logical Computer Store
1067 Farmington Ave.
Farmington, CT 06032
(203) 674-8405

Greenwich
GKG Enterprises,
Microage of Greenwich
280 Railroad Ave.
Greenwich, CT 06803
(203) 629-8171

Guilford
Bright Ideas Creative
Computer Center
1310 Boston Post Rd.
Guilford, CT 06437
(203) 453-6665

Hamden
Computerland, North Haven
60 Skiff St.
Hamden, CT 06517
(203) 288-5162

West Hartford
Computerland
131 S. Main St.
West Hartford, CT 06107
(203) 561-1446

Milford
Computerease Inc.
232 Boston Post Rd.
Milford, CT 06460
(203) 874-7447

New Haven
Computer City
377 Temple St.
New Haven, CT 06510
(203) 562-7546

Diversified Electric and Yale Coop
77 Broadway
New Haven, CT 06520
(203) 397-2607

New London
Computer Lab of Connecticut
531 Broad St.
New London, CT 06320
(203) 447-1079

Norwalk
Microcomputer Store Inc.
345 Main Ave.
Norwalk, CT 06851
(203) 846-4438

Southington
AM Computer Products
5 N. Main St.
Southington, CT 06489
(203) 621-2331

Stamford
Computer Factory
21 Atlantic St.
Stamford, CT 06901
(203) 356-1920

Computerland Stamford
111 High Ridge Rd.
Stamford, CT 06905
(203) 964-1224

Exel
2235 Summer St.
Stamford, CT 06905
(203) 348-5894

Intro II Inc.
36 Atlantic St.
Stamford, CT 06901
(203) 964-1133

West Hartford
Computer City
1475 New Britain Ave.
West Hartford, CT 06107
(203) 521-2245

West Haven
The Computer Mall
1177 Orange Ave., U.S. Rte. 1
West Haven, CT 06516
(203) 933-1541

Westport
Computer Works
1439 Post Rd. E
Westport, CT 06880
(203) 255-9096

Wethersfield
Computer Resources
683 Silas Deane Hwy.
Wethersfield, CT 06109
(203) 563-9000

DISTRICT OF COLUMBIA

Washington, DC
Computers Plus
1729 DeSales St. NW
Washington, DC 20036
(202) 331-9030

The Computer Store
1990 K St. NW
Washington, DC 20006
(202) 466-3367

Computerland of Farragut Square
1620 I St. NW
Washington, DC 20006
(202) 835-2200

The Math Box
2109 L St. NW
Washington, DC 20037
(202) 463-7474

DELAWARE

Newark
Computerland, New Castle
232 Astro Shopping Ctr.—
Kirkwood Hwy.
Newark, DE 19711
(302) 738-9656

Wilmington
The Computer Store
4010 Concord Pike—Rte. 202N
Wilmington, DE 19803
(302) 478-7497
Computerland of North Wilmington
2308 Concord Pike
Wilmington, DE 19803
(302) 652-1300

FLORIDA

Altamonte Springs
Computerland
237 W. Hwy. 436
Altamonte Springs, FL 32701
(305) 862-6202

Boca Raton
Computer Center of Boca Raton
2200 Glades Rd., Ste. 602
Boca Raton, FL 33431
(305) 391-7300
Computerland
500 E. Spanish River Blvd.
Boca Raton, FL 33432
(305) 368-1122

Clearwater
Computer Works of Florida
2232 E. Bay Dr.
Clearwater, FL 33516
(813) 536-2737

Computerland
3485 U.S. 19 N
Clearwater, FL 33515
(813) 785-5579

HTS Computers
21 Clearwater Mall
Clearwater, FL 33516
(813) 796-1195

Ray's Computer Center
1590 U.S. Hwy. 19 S
Clearwater, FL 33516
(813) 535-1416

Coral Gables
Computerland
274 Alhambra Cir.
Coral Gables, FL 33134
(305) 442-4112

International Computer Systems
2201 Ponce De Leon Blvd.
Coral Gables, FL 33134
(305) 448-5960

Deland
Mini-Concepts #2
110 W. Rich Ave.
Deland, FL 32720
(904) 736-6464

Fort Lauderdale
Byte Shop
2176 W. Oakland Park Blvd.
Fort Lauderdale, FL 33311
(305) 486-2983

Computer Systems
7120 N. University Dr.
Fort Lauderdale, FL 33321
(305) 722-2220

The Computer Works
6221 N. Federal Hwy.
Fort Lauderdale, FL 33308
(305) 491-8600

Computerland, Fort Lauderdale
3963 N. Federal Hwy.
Fort Lauderdale, FL 33308
(305) 566-0776

Fort Meyers
Computerland
5673 S. Tamiami Tr.
Fort Meyers, FL 33907
(813) 939-7800

Micro Works
9600 S. Tamiami Tr. #134
Fort Meyers, FL 33907
(813) 936-4676

Fort Walton Beach
Grice-Fort Walton Beach
417-A Mary Ester Cutoff
Fort Walton Beach, FL 32548
(904) 244-3168

Gainesville
Computer System Resource
3222 S.W. 35th Blvd., Butler Plz.
Gainesville, FL 32604
(904) 376-4276

Gulf Breeze
Computer Store, Gulf Breeze
Gulf Breeze Pkwy.
Gulf Breeze, FL 32561
(904) 932-0660

Hialeah
Southern Microcomputer
15945 N.W. 57th Ave.
Hialeah, FL 33014
(305) 621-4137

Holly Hill
Mini-Concepts
1678 Northridgewood Ave., Ste. E
Holly Hill, FL 32017
(904) 673-1835

Hollywood
Computerdesign
2554 N. State Rd. 7
Hollywood, FL 33021
(305) 983-2300

Jacksonville
Computerland, Jacksonville
2777-6 University Blvd. W
Jacksonville, FL 32217
(904) 731-2471

Komputer Kingdom
10736 Atlantic Blvd.
Jacksonville, FL 32211
(904) 641-9042

SEB Computers
1705 University Blvd. N
Jacksonville, FL 32211
(904) 743-7050

Williams Radio & TV
2062 Liberty St.
Jacksonville, FL 32206
(904) 358-3707

Lakeland
Computers Etcetera Inc.
105 S. Fern Rd.
Lakeland, FL 33801
(813) 686-4310

Lakeland
Computerland
3320 S. Florida Ave.
Lakeland, FL 33803
(813) 644-6437

Maitland
Alpha Computer Center
100 Lake Ave.
Maitland, FL 32751
(305) 645-5522

Melbourne
H.I.S. Computermation
1295 Cypress Ave.
Melbourne, FL 32935
(305) 254-9399

Merritt Island
Creative Computer Services
2092 N. Courtenay Pkwy.
Merritt Island, FL 32952
(305) 453-5353

Miami
Burdines
22 E. Flagler St.
Miami, FL 33101
(305) 662-3308

Byte Shop, Miami
7873 Bird Rd.
Miami, FL 33155
(305) 264-2983

Computer Facts Inc.
7104 N.W. 72nd Ave.
Miami, FL 33166
(305) 888-6000

Computer Scene
1641 N.E. 163rd St.
Miami, FL 33162
(305) 945-1014

Computer Village
931 S.W. 87th Ave.
Miami, FL 33174
(305) 266-5965

Kendall Computer
10471 N. Kendall Dr.
Miami, FL 33176
(305) 274-6312

Naples
Micro Works Inc.
839 5th Ave. S
Naples, FL 33940
(813) 261-8151

Nokomis
Computer Center
909 S. Tamiami Tr.
Nokomis, FL 33555
(813) 484-1028

North Miami Beach
Computerland
760 N.E. 167th St.
North Miami Beach, FL 33162
(305) 944-9699

North Palm Beach
Computer Center of the
Palm Beaches
751 Northlake Blvd.
North Palm Beach, FL 33408
(305) 848-3801

Ocala
Personal Computer Center
3249 E. Silver Springs Blvd.
Ocala, FL 32670
(904) 726-1047

Panama City
Computer Center of Panama City
211-A W. 15th St.
Panama City, FL 32401
(904) 763-8038

Pensacola
Grice Electronics
6330 N. Pensacola Blvd.
Pensacola, FL 32505
(904) 477-8100

Gulf Coast Computer
4801 N. 9th Ave.
Pensacola, FL 32503
(904) 478-4730

Plantation
Computer Generation
7003 W. Broward Blvd.
Plantation, FL 33317
(305) 791-4578

Port Richey
F.M.S. Computers
1000-23 U.S. Hwy. 19
Port Richey, FL 33568
(813) 848-8121

Rockledge
Modern Computing Inc.
1275 S. U.S. #1
Rockledge, FL 32955
(305) 636-5279

Sarasota
Computerland
7374 S. Tamiami Tr.
Sarasota, FL 33581
(813) 923-4717

Seminole
Computerland
9430 Seminole Blvd.
Seminole, FL 33542
(813) 392-0771

Saint Petersburg
Executive Computer Systems Inc.
5018-66th St. N
Saint Petersburg, FL 33709
(813) 541-7711

Tallahassee
Business Computer Center
3425 Thomasville, Carriage Gate Ct.
Tallahassee, FL 32308
(904) 893-6969

Computerland
1815-6 Thomasville Rd.
Tallahassee, FL 32303
(904) 224-9341

Tampa
AMF Microcomputer Center
11612 N. Dale Mabry Hwy.
Tampa, FL 33618
(813) 963-1504

Computerland
1520 E. Fowler Ave
Tampa, FL 33612
(813) 971-1680

Florida Computer Center
800 Franklin St. Mall
Tampa, FL 33602
(813) 229-7335

Microcomputer Systems
144 S. Dale Mabry Hwy.
Tampa, FL 33609
(813) 875-0406

Winter Park
Atlantic Business Computers Inc.
1909 Aloma Ave.
Winter Park, FL 32792
(305) 677-7585

West Palm Beach
Computerland
4275 Okeechobee Blvd.
West Palm Beach, FL 33409
(305) 684-3338

Microage Computer Store
1683 Forum Pl.
West Palm Beach, FL 33401
(305) 683-5779

GEORGIA

Albany
Business Computer Center
2416 Dawson Rd.
Albany, GA 31701
(912) 888-2097

The Computer Store
404 Sands Dr.
Albany, GA 31705
(912) 883-2121

Athens
Computerland, Athens
Athena Sq.
Athens, GA 30604
(404) 548-5263

Atlanta
ACM Computer Mart
5091-B Buford Hwy.
Atlanta, GA 30340
(404) 455-0647

Advance Computer Technologies
6540 Roswell Rd.
Atlanta, GA 30328
(404) 255-8984

Computerland, Atlanta
2423 Cobb Pkwy.
Atlanta, GA 30080
(404) 953-0406

Computerland of Atlanta
3222 Peachtree Rd. NE
Atlanta, GA 30305
(404) 237-0102

Compushop of Georgia #3
2710 New Springs Rd. NW
Atlanta, GA 30339
(404) 998-9120

Compu Shop of Georgia
5600 Roswell NE, The Prado
Atlanta, GA 30342
(404) 252-9611

Peachtree Computer Center
231 Peachtree St.,
NE Shopping Gallery
Atlanta, GA 30303
(404) 522-0082

Southeastern Computer Center
3623 Interstate 85 N
Atlanta, GA 30340
(404) 457-8465

Augusta
Cash Register Exchange Inc.
1501 Hicks St.
Augusta, GA 30904
(404) 724-1747

Carrollton
Carrollton Office Equipment Co.
104 Pine Knoll Dr.
Carrollton, GA 30117
(404) 834-4436

Chamblee
Compushop of Georgia Inc.
3350-B Chamblee Tucker Rd.
Chamblee, GA 30341
(404) 451-5498

Columbus
Columbus Computer Centers Inc.
1165 Henry Ave.
Columbus, GA 31906
(404) 324-0667

Team Electronics
Columbus Sq. Mall
Columbus, GA 31906
(404) 568-0450

Gainesville
Preferred Audio
2341 Thompson Bridge Rd.
Gainesville, GA 30501
(404) 532-3857

Macon
Georgia Computer Products
147 Spring St.
Macon, GA 31201
(912) 741-1833

Marietta
Friendly Computers Limited
1255-11 Johnson Ferry Rd.
Marietta, GA 30067
(404) 973-1885

Morrow
Southside Computer Ctr.
1364 Morrow Industrial Blvd.
Morrow, GA 30260
(404) 968-4873

Savannah
Logical Choice
105 Montgomery Crossroads
Savannah, GA 31406
(912) 927-1699

Thomasville
Ponders Inc.
117 W. Jefferson St.
Thomasville, GA 31792
(912) 226-3341

Tifton
The Edwards Datasystems Inc.
402 N. Central Ave.
Tifton, GA 31794
(912) 386-1234

MASSACHUSETTS

Boston
The Computer Store Inc.
103 Devonshire St
Boston, MA 02109
(617) 426-4385

Computerland of Boston II
60 Congress St., P.O. Off Sq.
Boston, MA 02109
(617) 482-6033

Ferranti-Dege Inc.
455 Brookline Ave.
Boston, MA 02115
(617) 232-2550

Braintree
Hammetts Learning World
South Shore Plz., 250 Granite St.
Braintree, MA 02184
(617) 848-4096

Brookline
Metropolitan Computers
110 Harvard St.
Brookline, MA 02146
(617) 277-5115

Burlington
Computer City
Vinebrook Plz.
Burlington, MA 01803
(617) 273-3146

The Computer Store
120 Cambridge
Burlington, MA 01803
(617) 272-8770

Cambridge
Computer City
991 Massachusetts Ave
Cambridge, MA 02138
(617) 491-4638

The Computer Store
1678 Massachusetts Ave.
Cambridge, MA 02138
(617) 354-4599

Harvest Computer
118 A Magazine St.
Cambridge, MA 02139
(617) 547-3289

Harvard Cooperative Society
1400 Massachusetts Ave.
Cambridge, MA 02238
(617) 492-1000

Tech Computer Store
199 Alewife Brook Pkwy.
Cambridge, MA 02140
(617) 497-0395

Charlestown
Computer City
175 Main St.
Charlestown, MA 02129
(617) 242-3350

Computer City
420 Rutherford Ave
Charlestown, MA 02129
(617) 242-4596

Chicopee
Custom Electronics Inc.
238 Exchange St
Chicopee, MA 01013
(413) 592-4761

Danvers
Computer City
151 Endicott St.
Danvers, MA 01923
(617) 774-7118

East Longmeadow
The Small Computer Co.
10 Center Sq.
East Longmeadow, MA 01028
(413) 525-6663

Framingham
Computer City
50 Worcester Hwy.
Framingham, MA 01701
(617) 875-8126

The Computer Store
680 Worcester Rd.
Framingham, MA 01701
(617) 879-3720

Hanover
CPU Computer Center
Hanover Mall Plz.
Hanover, MA 02339
(617) 826-8213

Hyannis
S and E Computer Inc.
1019 Iyanough Rd.
Hyannis, MA 02601
(617) 775-6071

Two-Way Radio Service
358 W. Main St.
Hyannis, MA 02601
(617) 775-8176

Leominster
Computer Systems and Software
46 Mechanic St.
Leominster, MA 01453
(617) 537-1202

Ludlow
Northeast Computer Stores
455 Center St.
Ludlow, MA 01056
(413) 589-0106

Needham
New England Electronics Co.
679 Highland Ave.
Needham, MA 02194
(617) 449-1760

North Dartmouth
Computer 'N Things
21 Faunce Corner Rd.
North Dartmouth, MA 02747
(617) 997-0783

Northampton
Northeast Computer Stores
22 Masonic St.
Northampton, MA 01060
(413) 586-8845

Pittsfield
Computer Source
Berkshire Common, South St.
Pittsfield, MA 01201
(413) 443-7181

Reading
Computerland of Boston, North
343 Main St.
Reading, MA 01867
(617) 942-0707

Saugus
Land of Electronics
216 Broadway
Saugus, MA 01906
(617) 581-3133

Springfield
Market Place Computer
75 Market St.
Springfield, MA 01103
(413) 737-2002

Sudbury
The Computer Store
55 Union Ave.
Sudbury, MA 01776
(617) 879-3700

Taunton
Micro Store
9 Cape Rd.
Taunton, MA 02780
(617) 823-8106

Watertown
Microsource
23 Elm St.
Watertown, MA 02172
(617) 924-5500

Wellesley
Byrne Computer Center
5 Cameron St.
Wellesley, MA 02181
(617) 431-7010

Computerland, Boston
214 Worcester St.
Wellesley, MA 02181
(617) 235-6252

Sherman Howe Computer Centers
45 William St.
Wellesley, MA 02181
(617) 237-2122

Winthrop
Harbor Electronics
365 Main St.
Winthrop, MA 02152
(617) 846-7132

Woburn
Northeast Computer Stores
400 W. Commings Park, Ste. 1190
Woburn, MA 01801
(617) 935-8060

Worcester
Computer City
16 Front St.
Worcester, MA 01608
(617) 755-5464

The Computer Place
11 Harvard St.
Worcester, MA 01609
(617) 755-5387

MARYLAND

Annapolis
Computer Etc.
257 W. St.
Annapolis, MD 21401
(301) 268-5801

Computerland
West St./Rte. 2—Parole Sta.
Annapolis, MD 21401
(301) 266-6277

Baltimore
Balance Computer Center
17 S. Charles St.
Baltimore, MD 21201
(301) 625-1100

The Logical Choice
8 N. Calvert St.
Baltimore, MD 21202
(301) 625-1050

Beltsville
HLA Inc.
5700-J Sunnyside Ave.
Beltsville, MD 20705
(301) 345-1123

Bethesda
Bethesda Computers
8020 Norfolk Ave.
Bethesda, MD 20014
(301) 657-1992

Clinton
Clinton Computer Center
6443 Old Alexander Ferry Rd.
Clinton, MD 20735
(301) 868-0002

Cumberland
Miller and Miller Data Processing
49 N. Centre St.
Cumberland, MD 21502
(301) 777-1000

Easton
The Computer Shop
10 W. Dover St.
Easton, MD 21601
(301) 822-3200

Ellicott City
The Logical Choice #4332
9200 Baltimore Nat'l Pike
Ellicott City, MD 21043
(301) 465-3175

Frederick
Frederick Computer Products
5726 Industry Ln.
Frederick, MD 21701
(301) 694-8884

Hagerstown
Custom Computing Inc.
806 Frederick St.
Hagerstown, MD 21740
(301) 739-6500

Laurel
The Comm Center
9624 Ft. Meade Rd.
Laurel, MD 20707
(301) 953-9535

Lutherville
Computerland of Towson
1516 York Rd.
Lutherville, MD 21093
(301) 337-5555

Pikesville
The Logical Choice
1700 Reisterstown Rd.
Pikesville, MD 21208
(301) 653-3410

Rockville
The Computer Workshop
1776 E. Jefferson
Rockville, MD 20852
(301) 468-0455

Computerland of Rockville
451 Hungerford Dr., Ste. 109
Rockville, MD 20850
(301) 340-8484

The Math Box
1800 Rockville Pike
Rockvile, MD 20852
(301) 984-7717

Salisbury
The Computershop
112 W. Market St.
Salisbury, MD 21801
(301) 543-8200

Silver Spring
Computers Etc.
9330 Georgia Ave.
Silver Spring, MD 20910
(301) 588-3748

Towson
Computers Etc.
13A Allegheny Ave.
Towson, MD 21204
(301) 296-0520

Computers Unltd. Inc.
907 York Rd.
Towson, MD 21204
(301) 321-1553

Towson Computer
409 Washington Ave.
Towson, MD 21204
(301) 337-2750

Westminster
Westminster Computer Center
330 140 Village Rd.
Westminster, MD 21157
(301) 875-2637

Wheaton
Computer Crafters
11246 Georgia Ave.
Wheaton, MD 20902
(301) 933-5820

MAINE

Brewer
Northeast Computer Stores
Brewer Shopping Ctr.
Brewer, ME 04412
(207) 989-7563

Brunswick
Coastal Computer Center
149 Main St.—The Tontine Mall
Brunswick, ME 04011
(207) 729-0157

Lewiston
Advantage Business Store
208 Lisbon St.
Lewiston, ME 04240
(207) 783-1003

Portland
Computerland
84 Exchange St.
Portland, ME 04101
(207) 774-1309

Harper Electronics
114 Commercial St.
Portland, ME 04101
(207) 772-1156

Waterville
Home Port Computers Inc.
129 Main St.
Waterville, ME 04901
(207) 873-2192

MISSISSIPPI

Columbus
Business Equipment and Supply Co.
Hwy. 45 N
Columbus, MS 39701
(601) 328-6860

Greenville
Penn's Computer Center
234 Washington Ave.
Greenville, MS 38701
(601) 332-2671

Gulfport
Datalab
7F Norwood Shopping Village
Gulfport, MS 39503
(601) 832-8766

Hattiesburg
Computer World of Mississippi
2101 Hardy St.
Hattiesburg, MS 39401
(601) 544-3135

Jackson
Computerland
4328 N. State St.
Jackson, MS 39206
(601) 362-8754

Mississippi Micros Inc
Mart 51, 1700 Terry Rd.
Jackson, MS 39204
(601) 948-7846

Meridian
Williams Electronics
3304 20th St.
Meridian, MS 39301
(601) 483-7191

Natchez
Miss-Lou Computer Center
624 Franklin St.
Natchez, MS 39120
(601) 442-2836

Starkville
Teletronics
500 Russell St.
Starkville, MS 39759
(601) 323-4614

Tupelo
The Coming Attraction
420 S. Gloster
Tupelo, MS 38801
(601) 841-1236

NORTH CAROLINA

Apex
Surveyor's Supply Co.
Hwy. 64 at Old Hwy. 1
Apex, NC 27502
(919) 362-7000

Asheville
Computer Alternatives
Turtle Creek Shopping Ctr.
Asheville, NC 28803
(704) 274-5404

The Electronic Office
780 Hendersonville Rd.
Asheville, NC 28803
(704) 274-1196

Burlington
Carolina Biological Supply Co.
2700 York Rd.
Burlington, NC 27215
(919) 584-0381

Charlotte
The AMS Computer Store
6631 Morrison Blvd.
Charlotte, NC 28211
(704) 366-8492

Computer Room
4231 Monroe Rd.
Charlotte, NC 28205
(704) 377-9821

Computer South of Charlotte
4416 Central Ave.
Charlotte, NC 28205
(704) 567-6091

Computerland, Charlotte
3915 E. Independence Blvd.
Charlotte, NC 28205
(704) 536-8500

Standard Computers
3827 Wilkinson Blvd.
Charlotte, NC 28208
(704) 399-0228

Durham
Computerland of Durham
4125 Chapel Hill Blvd.
Durham, NC 27707
(919) 493-5402

Personal Computer Center
2605 Chapel Hill Blvd.
Durham, NC 27707
(919) 493-5466

NORTH CAROLINA

Fayetteville
Southeast Professional Computer
3358 Village Dr.
Fayetteville, NC 28304
(919) 323-2067

Gastonia
The Computer Workshop
1483-A Akers Shopping Ctr.
Gastonia, NC 28052
(704) 861-9030

Greensboro
Byte Shop, Greensboro
218 N. Elm St.
Greensboro, NC 27401
(919) 275-2983

Computer Shoppe
615 Guilford, Jamestown Rd., I40
Greensboro, NC 27409
(919) 299-4843

Greenville
Computer Displays
Greenville Sq. Shopping Ctr.
Greenville, NC 27834
(919) 756-9378

Hickory
Computer Alternatives
2153 Hwy. 64—70 E. Startown Plz.
Hickory, NC 28601
(704) 324-2040

High Point
Data Concepts
132 E. Parris Ave.
High Point, NC 27260
(919) 869-6717

Jacksonville
The Computerware Store
210 Henderson Dr.
Jacksonville, NC 28540
(919) 346-8499

Monroe
Hayes Computer Systems
313 N. Main St.
Monroe, NC 28110
(704) 289-4080

Raleigh
Audio Buys
1700 Glenwood Ave.
Raleigh, NC 27608
(919) 821-1776

Carolina Computer Store
2245 New Hope Church Rd.,
Brentwood Sq. Shop
Raleigh, NC 27604
(919) 876-5555

Wilmington
Computer Alternatives
4212 Oleander Dr.
Wilmington, NC 28403
(919) 799-5440

Winston-Salem
Computer South of Winston-Salem
8013 Northpoint Blvd.
Winston-Salem, NC 27106
(919) 748-8001

Computer Tree
1409-A Plaza West Rd.
Winston-Salem, NC 27103
(919) 768-9820

NEW HAMPSHIRE

Concord
Bitsnbytes Computer Center
56-B Pleasant St.
Concord, NH 03301
(603) 224-8233

Hanover
Chips Microcenter
3 South St.
Hanover, NH 03755
(603) 643-5413

Keene
Micro Services of New England
Riverside Plz.
Keene, NH 03431
(603) 357-1113

Manchester
Computer City
1525 S. Willow St.
Manchester, NH 03103
(603) 668-9527

Computerland
Livingston Park Plz.,
D. Webster Hwy. N
Manchester, NH 03104
(603) 668-2110

Nashua
Computer Mart of New Hampshire
170 Main St.
Nashua, NH 03060
(603) 883-2386

Computerland
419 Amherst
Nashua, NH 03063
(603) 889-5238

Plaistow
NEBA Computer Corp.
Rte. 125
Plaistow, NH 03865
(603) 382-4711

Plymouth
North Country Computer Center
Village Sq., Tenney Mt. Hwy.
Plymouth, NH 03264
(603) 536-4163

Portsmouth
Portsmouth Computer Ctr.
31 Raynes Ave.
Portsmouth, NH 03801
(603) 431-7438

Salem
Computer City
527 S. Broadway
Salem, NH 03079
(603) 898-2390

Computer Town
304 S. Broadway, Rte. 3
Salem, NH 03079
(603) 893-8812

NEW JERSEY

Bayonne
Parts Unltd. of Bayonne
433 Broadway
Bayonne, NJ 07002
(201) 339-5009

Boonton
World of Computers Inc.
516 Main St.
Boonton, NJ 07005
(201) 335-2342

Cherry Hill
The Computer Workshop
1200 Haddonfield Rd.
Cherry Hill, NJ 08002
(609) 665-4404

Computerland of Cherry Hill
1442 E. Rte. #70
Cherry Hill, NJ 08034
(609) 795-5900

East Hanover
Computer Mart of New Jersey
321 Rte. 10
East Hanover, NJ 07936
(201) 428-0200

Eatontown
Computerland, Eatontown
288 Rte. 35
Eatontown, NJ 07724
(201) 389-2333

Englewood
American Business Products
155 N. Dean St.
Englewood, NJ 07631
(201) 569-0853

Englishtown
Computer Madness
Summerton Shopping Plz., Rte. 9 S
Englishtown, NJ 07726
(201) 462-9696

Greenbrook
Computer Mart of New Jersey
229G Rte. 22
Greenbrook, NJ 08812
(201) 752-6300

Iselin
Computer Mart of New Jersey
501 Rte. 27
Iselin, NJ 08830
(201) 283-0600

Lawrenceville
Computer Mart of New Jersey
150 Mercer Mall
Lawrenceville, NJ 08648
(609) 452-1858

Computerland of Princeton
2940 Brunswick Pike
Lawrenceville, NJ 08648
(609) 882-1400

Marlton
Jonathan's Apple—Marlton
444 W. North Rte. 70
Marlton, NJ 08053
(609) 983-0668

Matawan
Vista Computer Shop
Pine Valley Plz., Hwy. 34
Matawan, NJ 07747
(201) 566-6066

Montvale
The Computer Center
112 Chestnut Ridge Rd.
Montvale, NJ 07645
(201) 391-1006

Morristown
Computerland of Morristown
74 Elm St.
Morristown, NJ 07960
(201) 539-4077

North Plainfield
Boise Office Equipment
136 Somerset St.
North Plainfield, NJ 07060
(201) 755-5544

Northfield
The Computer Port
1921 New Rd.
Northfield, NJ 08225
(609) 927-3880

Computerland of Northfield
Silo Shopping Ctr.
Northfield, NJ 08225
(609) 646-6611

Ocean
Transnet Corp.
1111 Rte. 35
Ocean, NJ 07712
(201) 531-7020

Paramus
The Computer Universe
155 Rte. 17
Paramus, NJ 07652
(201) 262-0960

Computerland, Bergen County
35 Plaza, Rte. 4 W
Paramus, NJ 07652
(201) 845-9303

Pine Brook
The Computer Nook
Pine Brook Plz., Rte. 46
Pine Brook, NJ 07058
(201) 575-9468

Pompton Plains
Computer Corner of New Jersey
439 Rte. 23
Pompton Plains, NJ 07444
(201) 835-7080

Princeton
Clancy Paul Inc.
Princeton Shopping Ctr.,
N. Harrison St.
Princeton, NJ 08540
(609) 683-0060

Computer Encounter
1225 Hwy. 206,
Princeton N. Shopping Ctr.
Princeton, NJ 08540
(609) 924-8757

Rutherford
Nubs Computer Center
6-6A Ames Ave.
Rutherford, NJ 07070
(201) 935-0012

Sea Girt
Brielle Computer Inc.
Fountain 9 Mall, Hwy. 35
Sea Girt, NJ 08750
(201) 449-8770

Sewell
Computer Professionals
Greentree & Egg Harbor Rds.
Sewell, NJ 08080
(609) 582-1234

Shrewsbury
Monmouth Computer Services
432 Broad St.
Shrewsbury, NJ 07701
(201) 747-6745

Stanhope
Computer Universe
23 Rte. 206
Stanhope, NJ 07874
(201) 347-7892

Stonehenge
Stonehenge Computer Shop
89 Summit Ave.
Stonehenge, NJ 07901
(201) 277-1020

Toms River
Atlantic Computer Systems
18 Washington St.
Toms River, NJ 08753
(201) 240-3101

Totowa
Computerland of Totowa
225 Rte. #46
Totowa, NJ 07512
(201) 785-4448

Union City
Adelman's Stationery
3223 Berganline Ave.
Union City, NJ 07087
(201) 867-2482

Union
Transnet Corp.
1945 Rte. 22
Union, NJ 07083
(201) 688-7800

Vineland
Jonathan's Apple—Vineland
419 S. Delsea Dr.
Vineland, NJ 08360
(609) 691-9111

Wayne
The Computer Pros Inc.
West Belt Mall
Wayne, NJ 07470
(201) 256-7070

NEW YORK

Albany
The Computer Room Ltd.
1492 Central Ave.
Albany, NY 12205
(518) 869-3818

Brooklyn
Wonderful World of Computers
7418 Fifth Ave.
Brooklyn, NY 11209
(212) 748-9696

Buffalo
Binary Orchard
S-5854 Camp Rd.
Buffalo, NY 14075
(716) 648-7167

Computer Systems Center
255 Delaware Ave.
Buffalo, NY 14203
(716) 842-1116

Modern Tek Shops Inc.
4498 Main St.
Buffalo, NY 14226
(716) 839-5800

Carle Place
Computerland of Nassau
79 Westbury Ave.
Carle Place, NY 11514
(516) 742-2262

Commack
Computerland of Suffolk
6181 Jericho Tnpk.
Commack, NY 11725
(516) 499-4484

Corning
Chemung Electronics Inc.
28 W. Market St.
Corning, NY 14830
(607) 962-4606

Depew
Computer Systems Center
4891 Transit Rd.
Depew, NY 14043
(716) 668-5998

Elmira
Chemung Electronics Inc.
601 E. Church St.
Elmira, NY 14901
(607) 733-8486

Endwell
Computer Tree
409 Hooper Rd.
Endwell, NY 13760
(607) 748-1223

Forest Hill
Computer Factory
100-17 Queens Blvd.
Forest Hill, NY 11375
(212) 896-0700

Fredonia
Bits & Bytes Computing
21 Water St.
Fredonia, NY 14063
(716) 673-1515

Garden City
Computer Factory
1301 Franklin Ave.
Garden City, NY 11530
(516) 248-6700

Glens Falls
The Computer Room
28 Ridge St.
Glens Falls, NY 12801
(518) 869-3818

Great Neck
Berliner Computer Center
668 Northern Blvd.
Great Neck, NY 11021
(516) 466-2700

Huntington Station
B C Communications
207 Depot Rd.
Huntington Station
Long Island, NY 11746
(516) 692-2735

Ithaca
Chemung Electronics Inc
210 N. Tioga St.
Ithaca, NY 14850
(607) 272-2225

Computerland
419 W. Seneca St.
Ithaca, NY 14850
(607) 277-4888

Johnson City
Computerland of Johnson City
580 Harry L Dr.
Johnson City, NY 13790
(607) 798-9306

Kingston
Computer Shop
207 Boices Ln
Kingston, NY 12401
(914) 336-8411

Latham
Castle Computers Inc.
720 New Loudon Rd.
Latham, NY 12110
(518) 783-9405

Little Neck
Computerland of Little Neck
251-07 Northern Blvd.
Little Neck, NY 11362
(212) 423-5280

Manhassett
Computer Microsystems
1196 Northern Blvd.
Manhassett, NY 11030
(516) 627-3640

Massena
Computerland of Messena
46 Main St.
Massena, NY 13662
(315) 769-9971

Melville
Future Visions Computer Store
70 Broad Hollow Rd.
Melville, NY 11747
(516) 423-7820

Middletown
Computer Emporium
37 North St.
Middletown, NY 10940
(914) 343-4880

Mount Kisco
The Computer Edge
41 S. Moger Ave.
Mount Kisco, NY 10549
(914) 666-6337

New Hyde Park
Berliner Computer Center
102 Jericho Tnpk.
New Hyde Park, NY 11040
(516) 775-4700

Niagara Falls
Turnkey Computer Services
1909 Main St.
Niagara Falls, NY 14305
(716) 285-5101

New Rochelle
Calculator & Computer Center
148A North Ave.
New Rochelle, NY 10801
(914) 576-3610

New York
Computer Era Corp.
1570 3rd Ave.
New York, NY 10028
(212) 860-0500

Computer Factory
485 Lexington Ave.
New York, NY 10017
(212) 687-5000

The Computer Store
40 E. 52nd St.
New York, NY 10022
(212) 832-2180

Computerland of New York
58 W. 44th St.
New York, NY 10036
(212) 840-3223

Datel Systems Corp.
1211 Ave. of the Americas
New York, NY 10036
(212) 921-0110

Future Data
95 Trinity Pl.
New York, NY 10006
(212) 732-3905

Harmony House II
1167 York Ave.
New York, NY 10021
(212) 751-9188

Interdynamics Data Systems
369 Lexington Ave.
New York, NY 10017
(212) 557-0180

Mc Graw Hill Bookstore
1221 Ave. of the Americas
New York, NY 10036
(212) 997-4100

Morris Decision Systems
70 Pine St.
New York, NY 10270
(212) 742-9590

Plattsburgh
Ucompute
96 Margaret St.
Plattsburgh, NY 12901
(518) 563-1679

Port Chester
A World of Computers
519 Boston Post Rd.
Port Chester, NY 10573
(914) 937-6662

Potsdam
Advanced Computer Systems
16 Market St.
Potsdam, NY 13676
(315) 265-5620

Poughkeepsie
ASD Office Systems
Rte. 55 & Titusville Rd.
Poughkeepsie, NY 12603
(914) 473-9400

Riverhead
Custom Computer Specialists
208 Roanoke Ave.
Riverhead, NY 11901
(516) 369-2199

Rochester
The Computer Center
344 E. Main St.
Rochester, NY 14604
(716) 262-3166

Computer Store, Rochester
2423 Monroe Ave.
Rochester, NY 14618
(716) 244-5000

Leon's Typewriter & Supply
103 Clinton Ave., S. I
Rochester, NY 14604
(716) 325-2787

Scarsdale
All Things Computer
686 White Plains Rd.
Scarsdale, NY 10583
(914) 723-6262

Schenectady
The Computer Shack
1594 State St.
Schenectady, NY 12306
(518) 382-1182

Staten Island
Compu-Micro's Computer Store
3255 Richmond Ave.,
Greenridge Plz.
Staten Island, NY 10312
(212) 948-2828

Syracuse
Computer Solutions
2716 Erie Blvd. E
Syracuse, NY 13224
(315) 445-0794

Omnifax
3216 Erie Blvd. E
Syracuse, NY 13214
(315) 446-1284

Vestal
Micro World Computer Store
1901 Vestal Pkwy. E
Vestal, NY 13850
(607) 785-4500

Wappingers Falls
Computer Systems Specialists
Imperial Plz., 118 Rte. 9
Wappingers Falls, NY 12590
(914) 297-1223

Watertown
Mc Dasny Computers
1222 Arsenal St.
Watertown, NY 13601
(315) 782-6808

White Plains
The Computer Corner
200 Hamilton Ave. Mall
White Plains, NY 10601
(914) 949-3282

The Computer Store
221 E. Post Rd.
White Plains, NY 10601
(914) 428-1661

Computerland of White Plains
600 N. Broadway
White Plains, NY 10603
(914) 328-0144

Whitesboro
Upstate Computer Shop
99 Commercial Dr., Rte. 5A, RD1
Whitesboro, NY 13492
(315) 768-8151

Yonkers
Computer Factory
2361 Central Park Ave.
Yonkers, NY 10710
(914) 793-1300

PENNSYLVANIA

Allentown
General Computer Store
1870 Airport Rd.
Allentown, PA 18103
(215) 266-1880

Allentown-Whitehall
Computerland, Lehigh Valley
1457 MacArthur Rd
Allentown-Whitehall, PA 18052
(215) 776-0202

Altoona
Juniata Office Supply
601 4th Ave. Juniata
Altoona, PA 16601
(814) 944-7291

Mace Electronics
3325 Pleasant Valley Blvd.
Altoona, PA 16603
(814) 942-5031

Bloomsburg
The Computer Clinic
1123 Old Berwick Rd.
Bloomsburg, PA 17815
(717) 387-1811

Butler
Mace Electronics #6
Alameda Plz./New Castle Rd.
Butler, PA 16001
(412) 282-3733

Chambersburg
Sunrise Electronics
2655 Philadelphia Ave.
Chambersburg, PA 17201
(717) 264-8214

Collegeville
Sun Information Management
323 Main St.
Collegeville, PA 19426
(215) 489-1911

Doylestown
Solution Computer Center
33 N. Main St.
Doylestown, PA 18901
(215) 345-4411

Dresher
Computerland of Dresher
1650 Limekiln Pike
Dresher, PA 19025
(215) 542-8835

Erie
Erie Computer Co.
2131 W. 8th St.
Erie, PA 16505
(814) 454-7652

PENNSYLVANIA

Erie
Mace Electronics
2631 W. 8th St.
Erie, PA 16505
(814) 838-3511

Mace Electronics
130 Millcreek Mall
Erie, PA 16505
(814) 868-4696

Fairless Hills
Computerware
203 Lincoln Hwy.
Fairless Hills, PA 19030
(215) 547-6700

Frazer
The Computer Forum
490 Lancaster Pike
Frazer, PA 19355
(215) 296-3474

Gibsonia
Computerland, Pittsburgh
5499 William Flynn Hwy
Gibsonia, PA 15044
(412) 433-0690

Grindstone
T J Enterprises
Rte. 40E
Grindstone, PA 15442
(412) 785-5311

Haverford
The Computer Store
25 Haverford Station Rd.
Haverford, PA 19041
(215) 642-9830

Hershey
The Office Works
241 W. Chocolate Ave.
Hershey, PA 17033
(717) 533-5900

Horsham
Mighty Byte Computer Centers
537 Easton Rd., Horsham Plz.
Horsham, PA 19044
(215) 443-9020

King of Prussia
Camerart
445 Goddard Blvd.
King of Prussia, PA 19406
(215) 337-2020

Computer Professionals
King of Prussia Plz.
King of Prussia, PA 19406
(215) 337-2525

Lancaster
Computerland of Lancaster
550 Centerville Rd.
Lancaster, PA 17601
(717) 898-3013

The Office Works
29 E. King St.
Lancaster, PA 17603
(717) 397-7721

Langhorne
Video Computer Center
Rte. 413 & Dblwds Rd.,
Summit Sq. Shopping Ctr.
Langhorne, PA 19047
(215) 860-0550

Ligonier
Computer Spot
On the Diamond
Ligonier, PA 15658
(412) 238-2381

Meadville
Mace Electronics
Meadville Mall, Conneaut Lake Rd.
Meadville, PA 16335
(814) 724-2880

Mechanicsburg
Computerland, Harrisburg
4644 Carlisle Pike
Mechanicsburg, PA 17055
(717) 763-1116

Monroeville
Computer Workshop, Pittsburgh
3848 William Penn Hwy.
Monroeville, PA 15146
(412) 823-6722

New Brighton
Television Parts Co.
518 5th Ave.
New Brighton, PA 15066
(412) 846-3000

Paoli
Computerland, Paoli
81 E. Lancaster Ave.
Paoli, PA 19301
(215) 296-0210

Philadelphia
Bundy Typewriter Co.
10th & Chestnut Sts.
Philadelphia, PA 19107
(215) 992-0500

Bundy Typewriter Co.
Roosevelt Mall, 2355 Cottman Ave.
Philadelphia, PA 19111
(215) 332-5600

Caldwell Computer Corp.
6720 Rising Sun Ave.
Philadelphia, PA 19111
(215) 742-8900

Intelligent Electronics
8th & Market Sts.
Philadelphia, PA 19105
(215) 629-6000

University Business Machines
3501 Chestnut St.
Philadelphia, PA 19104
(215) 387-5978

Pittsburgh
Business Equipment Sales
5284 Steubenville Pike
Pittsburgh, PA 15202
(412) 923-2533

The Computer House
333 Mansfield Ave.
Pittsburgh, PA 15220
(412) 921-1333

Computer Store of Pittsburgh
612 Smithfield St.
Pittsburgh, PA 15222
(412) 391-8050

Computerland, South Hills
429 Cochran Rd.
Pittsburgh, PA 15228
(412) 344-0690

Morgan's Computer and
Educational Center
535 Clairton Blvd.
Pittsburgh, PA 15236
(412) 653-6150

Reading
Computerland of Reading
833 N. Park Rd.
Reading, PA 19610
(215) 376-6500

Scranton
Kemor Data Systems
1702 Ash St.
Scranton, PA 18510
(717) 961-5888

Springfield
VR Data Computer Center
616 W. Baltimore Pike
Springfield, PA 19064
(215) 328-7300

State College
The Computer Store
1402 S. Atherton St.
State College, PA 16801
(814) 237-3444

West Reading
The Computer Source
546 Penn Ave.
West Reading, PA 19610
(215) 373-1264

Wayne
Mainline Computer Center
155 E. Lancaster Ave.
Wayne, PA 19087
(215) 687-8500

Willow Grove Park
Camerart
2500 Moreland Rd.
Willow Grove Park, PA 19090
(215) 328-5700

Williamsport
The Computer Clinic
138 W. 4th St.
Williamsport, PA 17701
(717) 546-5946

York
Computers Unltd.
2813 E. Prospect Rd.
York, PA 17402
(717) 755-1045

RHODE ISLAND

Providence
Computer City
165 Angell St.
Providence, RI 02906
(401) 331-2187

The Computer Store
740 N. Main St.
Providence, RI 02904
(401) 331-0220

Computerland of Providence
123 Dyer St.
Providence, RI 02903
(401) 274-5100

Unicorn Division of United Camera
297 Elmwood Ave.
Providence, RI 02907
(401) 467-5600

SOUTH CAROLINA

Anderson
Computerland of Anderson
121 Beltline Blvd.
Anderson, SC 29621
(803) 224-5428

Charleston
Computer Source
1660 Hwy. 7
Charleston, SC 29407
(803) 571-1452

Columbia
Byte Shop Columbia
2303 Devine St.
Columbia, SC 29205
(803) 771-7824

The Computer Store
810 Dutch Square Blvd.
Columbia, SC 29210
(803) 798-3300

Florence
Creative Computer Systems
1812 N. Irby St.
Florence, SC 29501
(803) 665-8655

Greenville
Computer Haven
469 Congaree Rd.
Greenville, SC 29607
(803) 297-6295

Datamart of South Carolina
1901 Laurens Rd.
Greenville, SC 29662
(803) 233-5753

Greenwood
Greenwood Computers Inc.
526 E. Durst Ave.
Greenwood, SC 29646
(803) 229-7575

Myrtle Beach
Creative Computer Systems
1204-1/2 N. Kings Hwy.
Myrtle Beach, SC 29577
(803) 665-8655

Rock Hill
Computerland of Rock Hill
2423 Cherry Rd.
Rock Hill, SC 29730
(803) 324-1121

Gismo Communications Inc.
2305 Cherry Rd.
Rock Hill, SC 29730
(803) 366-7157

Spartanburg
Apple Plus
451 Hillcrest Shopping Ctr.
Spartanburg, SC 29301
(803) 583-7766

TENNESSEE

Bristol
Rush Electronics Inc.
1315 Bluff City Hwy.
Bristol, TN 37620
(615) 764-0831

Chattanooga
Computerland of Chattanooga
5948 Brainerd Rd.
Chattanooga, TN 37421
(615) 892-0840

Peter Drew Computers
5475 Hixson Pike
Chattanooga, TN 37343
(615) 842-5813

Goodletsville
Computer Shoppe Inc.
The Village at Rivergate
Goodletsville, TN 37115
(615) 859-4015

Jackson
Computerlab
133 Old Hickory Blvd.
Jackson, TN 38201
(901) 668-9282

Kingsport
Computerland
2601 Fort Henry Dr.
Kingsport, TN 37664
(615) 246-6173

Joseph's Musicenter
128 Broad St.
Kingsport, TN 37660
(615) 246-4189

Knoxville
Computerland
8807 Kingston Pike
Knoxville, TN 37923
(615) 693-8225

Eastern Computer Inc
5613 Kingston Pike
Knoxville, TN 37919
(615) 584-8365

M C S Inc.
2831 Broadway
Knoxville, TN 37917
(615) 689-6601

Memphis
Computerlab of Memphis
5041 Park Ave.
Memphis, TN 38117
(901) 761-4743

Computerland
4840 Poplar Ave.
Memphis, TN 38117
(901) 767-0233

Cooper Office Equipment
121 Union Ave.
Memphis, TN 38103
(901) 526-3227

Executive Computer Center
1233 Park Place Ctr.
Memphis, TN 38119
(901) 682-2539

Opus 2
747 Brookhaven Cir.
Memphis, TN 38117
(901) 683-0117

Murfreesboro
Anderson Computers Inc.
915 N.W. Broad St.
Murfreesboro, TN 37130
(615) 896-1600

Nashville
The Computer Shoppe
3813 Hillsboro Rd.
Nashville, TN 37215
(615) 292-4496

The Computer Shoppe
5th and Church
Nashville, TN 37219
(615) 292-4496

Surya Data Systems
1016 8th Ave. S
Nashville, TN 37203
(615) 254-5085

Oakridge
Educational Computer System
136 Fairbanks Plz.
Oakridge, TN 37830
(615) 483-4915

Sevierville
SVGATA Computer Store
129 Forks of the River Pkwy.
Sevierville, TN 37862
(615) 523-0401

Tullahoma
Anderson Computers
930A N. Jackson St.
Tullahoma, TN 37388
(615) 455-7459

VIRGINIA

Alexandria
Computer Plus
6120 Franconia Rd.
Alexandria, VA 22310
(703) 922-7850

Universal Computers
1710 Fern St.
Alexandria, VA 22302
(703) 379-0367

Charlottesville
Computerland, Charlottesville
Shopper's World, Hwy. 29N
Charlottesville, VA 22901
(804) 973-5701

Universal Computers
1915-A Cedar Hill Rd.
Charlottesville, VA 22906
(804) 971-6771

Fairfax
The Math Box
9650 Main St.
Fairfax, VA 22030
(703) 978-5400

The Math Box
11011 Lee Hwy.
Fairfax, VA 22032
(703) 691-8600

Grafton
Chaney Computer Association Inc.
2360-B Geo. Wash. Mem. Hwy.
Grafton, VA 23692
(804) 898-3667

Harrisonburg
Computer Works
785 E. Market St.
Harrisonburg, VA 22801
(703) 434-1120

Leesburg
Computer Solutions
26A Plaza St.
Leesburg, VA 22075
(703) 777-3770

Lynchburg
The Computer People
10108 Timberlake Rd.
Lynchburg, VA 24502
(804) 237-6633

Mc Lean
The Computer Store
6858 Old Dominion Dr.
Mc Lean, VA 22101
(703) 821-8333

Newport News
(Home) Peninsula Computer Center Inc.
12588 Warwick Blvd.
Newport News, VA 23606
(804) 595-1955

Reston
Universal Computers of Reston
2355 Hunter's Wood Plz.
Reston, VA 22091
(703) 620-6160

Richmond
Computer Concepts
8207 Hermitage
Richmond, VA 23228
(804) 288-1426

Computer Techniques Inc.
9207 Midlothian Tnpk.
Richmond, VA 23235
(804) 320-7933

Computerland of Richmond
9772 Gayton Rd.
Richmond, VA 23233
(804) 741-3502

First Step Computers
2909 Hathaway Rd.
Richmond, VA 23225
(804) 320-6496

Roanoke
Business Solution Center
2501 Williamson Rd., Ste. #10
Roanoke, VA 24012
(703) 366-2911

Jack L Hartman and Co.
2840 Peters Creek Rd.
Roanoke, VA 24019
(703) 362-1891

Springfield
Computers Etc.
6671-19 Backlick Rd.
Springfield, VA 22105
(703) 644-5500

The Math Box
Springfield Plz., Keene Mill Rd.
Springfield, VA 22152
(703) 451-7100

Vienna
Computerland of Tyson's Corner
8411 Old Courthouse Rd.
Vienna, VA 22180
(703) 893-0424

Virginia Beach
Computerland of Virginia Beach
509 Bird Neck Rd.
Virginia Beach, VA 23452
(804) 422-8271

The Memory Bank
5760 N. Hampton Blvd.
Virginia Beach, VA 23455
(804) 481-7626

Small Business Computer Center
2927 Virginia Beach Blvd.
Virginia Beach, VA 23452
(804) 340-1977

Woodbridge
Computerland of Woodbridge
1305 Jefferson Plz.
Woodbridge, VA 22191
(703) 491-4151

VERMONT

Barre
Ormsby's Computer Store
61 N. Main St.
Barre, VT 05641
(802) 479-1271

Brattleboro
Datatronic Computer Center
Putney Rd.
Brattleboro, VT 05301
(802) 257-0555

Burlington
Computerland of Northern Vermont
214 College St.
Burlington, VT 05401
(802) 658-5858

Rutland
Electronic Applications
36 Merchants Row
Rutland, VT 05701
(802) 773-3145

Index

B

C

D

E

F

G

H

K

L

M

N

O

P

Q

R

S

T

U

V

W

X, Y, Z

Other Great DATAMOST Books

THE APPLE ALMANAC *by Eric Goez and Willliam B. Sanders*
This book is that one-of-a-kind reference almanac where you can look everything up in ONE place: POKEs, PEEKs, CALLs, Applesoft, Integer and DOS commands, sort algorithms, opcode addressing charts, DCT & FMPL charts. **THE APPLE ALMANAC** is packed with important programming information. *$19.95*

APPLESOFT ENCYCLOPEDIA *by Loy Spurlock*
The definitive reference for all commands, routines, functions, and tricks available to the Applesoft BASIC programmer. **THE APPLESOFT ENCYCLOPEDIA** combines the features of a ready-reference with the tutorial qualities of a how-to manual. This classic book is destined to become a standard in the industry. *$39.95*

USING 6502 ASSEMBLY LANGUAGE *by Randall Hyde*
Tailored for the Apple, this book teaches how to write programs in 6502 assembly language. Basic symbols and terminology, an introduction to computer concepts, simple instruction examples, and detailed 6502 instructions are written in a clear, thorough style. For the serious programmer! *$19.95*

p-SOURCE *by Randall Hyde*
A comprehensive guide to the Apple Pascal System. Learn programming techniques, internal operation of the p-machine, and how to interface peripherals and non-standard devices to Apple Pascal. Includes an appendix on ATTACH-BIOS. *$24.95*

SOFTWARE AUTHOR'S GUIDE *by Mildred Heiney*
The first comprehensive guide to microcomputer software publishers. Each entry includes name, address, phone number, contact person, computer type, plus indispensible sales and marketing information. Make sure your software is submitted in the proper format and is compatible with the publisher's line! *$19.95*

PROGRAMMING FOR PROFIT *by Don K. Crowther*
Aimed at the programmer who wants to get his or her software published. **PROGRAMMING FOR PROFIT** is a guidebook through the maze of traditions, rules and standards in the software industry. How to choose the right publisher, submit a proposal, negotiate a contract, what to expect in the way of royalties, advances, sales and much more. *$14.95*

APPLE HOME COMPANION *by George Beekman and Dennis Corliss*
What use is a computer? How can it help in the home? The **APPLE HOME COMPANION** has the answers! Written for *everyone*, this books discusses practical applications for the Apple: how to buy and take care of your computer, "hands on" lessons, BASIC programming, word processing, education, games, data bases, telecommunications, graphics, music and peripherals — explained in a humorous, friendly style. Reviews of current commercial software are also included to aid you in smart buying. *$19.95*

THE ELEMENTARY APPLE *by William B. Sanders*
Leads you step-by-step through the process of hooking up your Apple, loading and saving programs, creating graphics and sound, using handy utilities, even writing your own programs. Even if you know nothing about computers, this book will help you join the computer revolution. *$14.95*

Find these great books at your local computer dealer or bookstore.
Or order direct from:

DATAMOST™
20660 Nordhoff Street, Chatsworth, CA 91311-6152
(818) 709-1202

$29.95

APPLE THESAURUS

Apple Thesaurus is *the* comprehensive reference book on Apple computers. From the Apple I to peripherals for the Apple //c, every facet of Apple hardware and software is fully explained.

This valuable book contains information on:

• Processing and memory • Output, input and expansion • Hardware • Programming • The video display generator • 80 column text display • Graphics and color • Keyboards for character input • Touch screens, light pens, and digitizers • The mouse • Sound, music and voice • Apple speech synthesis • Electricity and the power supply • The electronic Apple • Switches, counters and converters • Control systems • Serial signals • Modems • Terminal and printer interfaces • Printers and plotters • Local area networks • RAM and ROM • Disk drive functions • Disk capacity • Bank switching • The 6502 chip • Other microprocessors and their families • Assemblers and debuggers • Sector I/O and file management • The Applesoft interpreter • and much more!

Organized in an easy-to-follow manner, and filled with clear illustrations and diagrams, *Apple Thesaurus* is sure to become *the* standard reference for all Apple II, II+, //e and //c users.

DATAMOST INC.™
20660 Nordhoff Street, Chatsworth, CA 91311-6152
(818) 709-1202

ISBN 0-88190-346-9