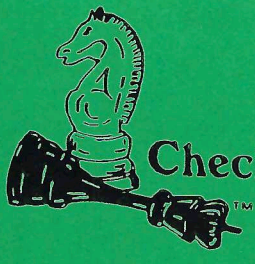


Checkmate Technology, Inc.™

MultiRam™ CX Kit

For the Apple //c

16-BIT MICROPROCESSING POWER

Interfaces with the MultiRam CX RAM card.

Checkmate Technology, Inc.

509 South Rockford Dr.
Tempe, Arizona 85281
(602) 966 5802

MULTIRAM CX KIT

A 65C816 16-bit Processor
for the
Apple //c Computer

Part #CX8165-1

TABLE OF CONTENTS

APPENDICES - 65C816 Specifications

READ ME FIRST

This preliminary manual WILL:

1. Tell you how to install the MULTIRAM 65C816 CX KIT on the MultiRam CX card in your Apple //c.
2. Explain how the 65C816 addresses the Apple //c's 128K memory plus the MultiRam CX's 512K added memory.
3. Provide 65C816 specifications and some 65C816 programming tips for assembly language programmers.
4. Refer you to existing and forthcoming 65C816 programming tools and informational materials.

This preliminary manual WILL NOT:

1. Instruct you in programming the 65C816. Although some tips will be offered, a complete programming guide is beyond the scope of this manual.
2. Detail all developments occuring with the 65C816. These developments are fast breaking and this manual will be outdated as soon as it is published.

A SPECIAL OFFER

This preliminary manual will be replaced with a more comprehensive manual in late 1985. In addition, a disk containing a 16-bit 65C816 operating system, called MAX-OS, and a set of utilities and demonstration software will be available at the same time.

You will automatically receive the new manual and disk by simply mailing in your warranty registration card.

VIP Professional, a Lotus 1-2-3 work-alike software package designed specifically for the MultiRam 65C816 cards, the first major application program for the 65C816, will be available in February of 1986. You can purchase VIP Professional for the 65C816 cards at a reduced price that will be offered during a limited period.

Refer to Chapter 6 for more information on both MAX-OS and VIP Professional.

Don't delay — send in your registration card today to be certain you are kept up to date.

FOR MULTIRAM C OWNERS

The 65C816 option from the CX Kit will work only on a MultiRam CX card. The option WILL NOT WORK on an earlier MultiRam C card.

If you own an older MultiRam C card, you can update it to a new MultiRam CX Card.

Simply take your old MultiRam C card along with $25.00 to a Checkmate Technology dealer and you will receive a new MultiRam CX Card with the same amount of memory as the old card.

You may also send your old MultiRam C card along with a $25.00 check or money order to Checkmate Technology and you will be shipped a new MultiRam CX Card with the same amount of memory as the old card.

The MultiRam CX card is identified by the words "MultiRam CX" appearing on the bottom center of the front of the card; the MultiRam C Card will be marked "MultiRam C". The CX card also has a component on the bottom right side of the front of card; the C Card does not. Other than the ability to use a 65C816, the two boards function identically.

This offer does not apply to those MultiRam C cards labeled "Not Upgradeable" on the board.

A //C WARNING

The MultiRam CX Card will not install onto the motherboard of some Apple //c's.

Installation of the MultiRam X Card requires the removal of the Memory Management Unit (MMU) component from the //c's motherboard. All older non-enhanced Apple //c's have a socketed MMU which can be easily removed from its socket. All //c's currently in production have a socketed MMU. Some Apples //c's manufactured in mid-1985, however, have the MMU soldered directly to the motherboard.

Your //c's MMU is socketed if it is fitted into a socket like the 65C02 CPU in front of it (refer to your MultiRam CX manual for location of the MMU and CPU on the motherboard). If the MMU is soldered onto the motherboard, you have three choices: (1) ask an authorized Apple dealer for a motherboard replacement, (2) have the MMU socketed, or (3) return the CX card and Kit to your dealer for a refund.

Adding a socket to your motherboard will void your warranty if your Apple is under initial 90 day warranty or if you have an extended care contract. You should contact your authorized Apple dealer regarding the installation of a socket if your Apple is under warranty. If your Apple is out of warranty, adding a socket can be done by a dealer or any qualified technician.

Chapter 1

INTRODUCTION

Welcome to the world of the 16-bit Apple //!

The MultiRam 65C816 CX Kit for the Apple //c and the compatible 65C816 MultiRam EX card for the Apple //e bring 16-bit processing power to the Apple // family of computers.

THREE COMPUTERS IN ONE

With the addition of the CX Kit's 65C816 to your Apple //c, your //c becomes three computers in one:

1. A standard //c capable of using all existing software and hardware peripherals.

2. A true 16-bit computer with computational speeds twice as fast (and more) than the normal //c with the added ability to address up to 16 megabytes of memory directly.

3. A 16-bit memory enhanced //c with up to 640K for running advanced 16-bit programs, or memory enhanced existing 8-bit programs, or for use as a lightning fast RAM disk.

The MultiRam CX's 65C816 fits directly onto the CX card which inserts into the //c motherboard completely replacing the //c's own 65C02. MultiRam CX's unique motherboard placement makes the 65C816 a integral part of the Apple ensuring both software and hardware compatibility.

MultiRam CX with the 65C816 turns your Apple into a very powerful personal computer, one suited for the most demanding personal and business uses. With the CX option, the 512K memory on the CX card becomes linearly and directly addressable memory.

You may have already experienced the power of a MultiRam memory enhanced //c by using AppleWorks with the expanded Desktop the MultiRam CX memory card provides AppleWorks, a Desktop of up to 425K.

In the near future, you will also be able to run new, powerful integrated programs that will not only use MultiRam's additional memory — as AppleWorks and other programs now do — but these programs will additionally take advantage of the advanced 65C816 CPU the CX Kit adds to your //c.

Checkmate Technology will offer the first of these programs, VIP Professional, a 16-bit Lotus 1-2-3 work-alike designed for the 65C816, early next year (see Chapter 5 for more details). Such programs mark the start of a whole new wave of 16-bit software for the Apple //c and //e.

65C816 ADVANTAGES

The 65C816 CPU, which makes advanced programs possible on your Apple, offers three advantages when used in your Apple //c:

65C02 Compatability

The 65C816 is two processors in one: a standard 8-bit 65C02 and a 16-bit 65C816.

The 65C816 is activated in what is called "65C02 emulation mode" when you first turn on the power to your //c.

In emulation mode, the 65C816 appears to all software to be an 8-bit 65C02 CPU, the same processor that is used in a standard Apple //c. The 65C816 can emulate the 65C02 because the commands of the 65C02 CPU are a subset of the 65C816's more extensive command set.

As a result, all software written for your //c will run on your new, more powerful Apple and all peripherals connected to it should work normally.

Until a simple software command is given to a special 65C816 status register bit called the Emulation Bit (E) , the 65C816 will continue to operate as a 65C02 CPU. Because the command to change the 65C816 into its native 16-bit mode can be given only from a special software command not available on the 65C02, existing 65C02 software cannot accidently shift you into 16-bit mode.

Using the simply addressed Emulation bit, the 65C816 can be easily toggled between operation in 8-bit and 16-bit mode. Due to its 100% compatibility with the 65C02 and easy shifting between 8-bit and 16-bit operation, developing software for the 65C816 is easier and should take less time than development of software for an entirely new processor for the Apple.

For example, by rewriting memory management and math or data manipulation routines of existing Apple programs into 16-bit 65C816 code while leaving other parts of programs untouched in 65C02 8-bit mode, significant performance improvements can be made relatively quickly to already existing software.

Faster Speed

The 65C816, when switched into its 16-bit native mode with a simple software command, has six 16-bit registers (Accumulator, X and Y indexes, Stack Pointer, Direct Page, and Program Counter registers) versus the 65C02's four 8-bit registers and one 16-bit Program Counter.

True internal 16-bit registers and operation from a 16-bit Arithmetic Logic Unit results in faster computation times. For example, math routines used in spreadsheets or search and sort routines common in databases can be speeded up by a factor of two to eight times over standard //c 65C02 routines depending upon the operation involved.

An additional benefit from 16-bit operation is that less instructions are required to perform the same computations than would be required with a 65C02. As a result of this efficiency, many routines can be reduced to approximately half the size of comparable 65C02 code. So 65C816 programs can be both faster and more compact than than 65C02 programs.

Zero page addresses (renamed direct addresses) and the stack can be defined in 16-bit rather than 8-bit values. Consequently, zero page and the stack can be placed anywhere in the first 64K of memory rather than in the fixed 256 byte locations where they normally exist in the standard //c. Because zero page and stack addressing is faster than absolute addressing, having more available zero pages (256 zero pages for the 65C816 versus one for the 65C02) and more stack locations can speed up a program.

New block move commands also speed up operations that require moving large amounts of data from one range of memory to another. The 65C816 can move large amounts of memory up to four or more times faster than the //c's standard 65C02. Faster moves are possible as built-in 65C816 move commands automatically move a range of memory given only source and destination addresses and the number of bytes to move.

For example, in less than half a second, 64K can be moved. The //c's standard 65C02, in comparison, requires a dedicated move routine to do what the 65C816's built-in commands do resulting in much slower speed.

The speed advantages of the 65C816 are derived from its more efficient internal operation. The CX's 65C816, because it is tied to the Apple's data bus to ensure greater software and hardware compatibility, runs at the Apple's effective data bus speed, approximately 1 megahertz.

16 Megabyte Direct Addressing

When in 16-bit native mode, the 65C816 can directly address 16 megabytes of memory as the 65C816 has a 24-bit address bus (a range from $00 0000 to $FF FFFF in hexadecimal notation). In comparison, the 65C02 CPU used in the //c can only address 64K of memory at a time as it has a 16-bit address bus ($0000 to $FFFF). In order for the 65C02 to address more than one block of 64K RAM, relatively slow, complex, and code intensive bank-switching techniques are required.

Therefore, when in 16-bit mode, the 65C816 on your MultiRam CX card can directly, easily and quickly address the 512K of memory on the MultiRam CX card and the 128K on the motherboard of the //c. This ability adds to the speed advantage of the 65C816 when a program must address more than 64K of program or data. Due to direct addressing versus bank-switched addressing for the 65C02, the 65C816 can transfer data through large ranges of memory up to four or more times faster than the 65C02.

Having large amounts of memory directly addressable allows more efficient as well as faster programs to be created. For example, spreadsheet, database, and word processor files can stretch throughout large ranges of memory and can be easily tracked with comparatively simple routines. Integrated programs can keep track of common areas of memory containing shared data using less code as 65C816 program modules can communicate with one another without the overhead involved with 65C02 bank-switching.

USING THE MULTIRAM 65C816

With the MultiRam 65C816 options, a wealth of untapped Apple power is now available. Because the 65C816 is a new processor, how quickly and easily you are able to put this power to work for you depends upon your programming skills.

Assembly language programmers will be the first who can take advantage of the 65C816. If you are familiar with 65C02 assembly language programming, you already know 70% of the opcodes available on the 65C816 as 65C02 opcodes are a subset of the 255 opcodes of the 65C816. Assemblers for the MultiRam 65C816, many of which are enhanced assemblers commonly used for the Apple, are available today (described in Chapter 5). Opportunities abound for assembly language programmers to create the applications programs and software tools of the Apple //'s future on today's MultiRam enhanced Apple //c.

Programmers in Basic, Pascal, and other high level languages will be the second to take advantage of the 65C816. As of the date of this preliminary manual, no 16-bit high level language for the 65C816 is currently available. However, by January of 1986 both a 65C816 Basic and 65C816 Pascal will be available for the MultiRam 16-bit cards with C, Forth, and other 65C816 languages and language versions soon to follow (see Chapter 5).

For nonprogrammers, there are no application programs ready to run on the 65C816 as of the writing of this preliminary manual. That will change by February of 1986 with the introduction of a significant 65C816 program.

Checkmate Technology will ship in February of 1986 a special 65C816 version of VIP Technologies Lotus 1-2-3 work-alike program, VIP Professional (see Chapter 5 for more detailed information). VIP Professional for the MultiRam 65C816 cards will take advantage of all of the memory on the MultiRam memory cards and will be written in 16-bit mode to take advantage of the more efficient 65C816 opcodes.

The result will be an integrated spreadsheet that will be as fast or faster than Lotus 1-2-3 on the IBM, will offer a Macintosh mouse interface and pull down windows for easy use, and will use commands and create files that are 100% compatible with Lotus 1-2-3. More information about this program and a special offer to you as an early buyer of the 65C816 option can be found in Chapter 5.

The MultiRam family of cards has been designed to bridge the gap between today's 6502 Apple and the Apple // of the future. VIP Professional is just the first of new, powerful 16-bit 65C816 programs for the MultiRam enhanced Apple //c and //e, programs that will rival or surpass comparable programs available on the IBM or Macintosh.

Checkmate Technology is committed to enhancing the power of the Apple // computers. To that end, we have and will continue to work with software developers to ensure the development of software for the 65C816 as we firmly believe the future of the Apple // family of computers lies with the 65C816 CPU and larger memory.

Chapter 2

INSTALLATION

This chapter contains all the information you will need to install and test your MultiRam 65C816 CX Kit.

If your MultiRam CX Card alreeady contains the 65C816 option, you may skip the installation sections and turn to the section in this chapter on testing the 65C816 option.

UNPACKAGING THE CX KIT

Check the contents of your CX Kit to be sure you have the following:

- This owners manual
- 65C816 CPU (larger of the two Kit components)
- 65C816 PAL (smaller of the two Kit components)
- A warranty registration card

Contact your dealer or Checkmate Technology if anything is missing.

You should fill out and return your warranty registration card as soon as possible. As explained in the Preface, you are immediately eligible for the new manual and disk containing the 16-bit operating system MAX-OS and 65C816 utilities if you simply send in your warranty card.

CX KIT CARE

Ground yourself before removing the 65C816 CPU or the 65C816 PAL from the antistatic foam containing them. Touch any metal object to discharge static electricity from your body; static electricity may damage these components.

Remove the two components from the antistatic foam. The larger of the two components is the 65C816 CPU and is marked as such. The smaller component is the 65C816 PAL.

Inspect the pins of the components to be certain they have not been bent or damaged in shipping. The pins on the bottom of the components should be gently straightened using needle nose pliers if they have been bent.

INSTALLING THE CX KIT

The 65C816 CPU and 65C816 PAL can be installed on the MultiRam CX Card while the MultiRam CX card is outside the //c or after the MultiRam CX card has been inserted inside the //c.

Before starting, be sure you read "A //c Warning" in "READ ME FIRST" at the start of this manual. Also before starting, be sure you have the "CX" version and not the "C" version of the MultiRam card. The CX Kit will not work on the "C" version. See "For MultiRam C Owners" in "READ ME FIRST" for more information.

Outside Installation

Place the MultiRam CX Card in front of you on a flat surface with the lettering on the card facing you. Leave the antistatic foam still covering the connector pins on the back of the card.

Identify the 65C816 CPU socket, the bottom empty socket of the card labeled "65C02 OR OPTIONAL 65C816." You will insert the 65C816 CPU into this socket (see Figure 2.1).

To insert the 65C816 CPU into the CPU socket, line up the pins of the 65C816 with the socket holes on the card. The notch on the 65C816 should face to the left toward the RAM on the card (see Figure 2.2a). First insert one side of the 65C816's pins into one side of the socket. Next, using an inward pushing motion, seat the pins on the other side of the 65C816.

If the pins do not easily seat, you may need to very slightly bend all pins on both sides of the 65C816 inward. Place the 65C816 on a flat surface on its side. Lift the 65C816 slightly up so that pressure can be uniformily applied to all pins along one side. Apply firm, even, gradually increasing pressure until you feel some small inward movement in the pins. DO NOT apply too much pressure or the pins will bend too far. Repeat this procedure on the opposite side. Retry inserting the 65C816. Repeat this procedure if the pins are still too far outside the socket to be moved into the socket holes.

Identify the 65C816 PAL socket. This socket labeled "OPTIONAL 16L8" is at the bottom right corner of the card and contains a jumper block (see Figure 2.1 and 2.2b). Pull off the jumber block with your fingers to expose the socket underneath. You will insert the 65C816 PAL into this socket. Store the jumper block safely away.

To insert the 65C816 PAL, line up the pins of the PAL with the socket holes on the card. The notch on the PAL should face the top of the card (see Figure 2.2a). First insert one side of the PAL's pins into one side of the socket. Next using an inward pushing motion, seat the pins on the other side of the PAL.

If the pins do not easily seat, follow the previous instructions given for inserting the 65C816 with the same problem.

If you remove the 65C816 CPU and 65C816 PAL for any reason and insert the //c's own 65C02 onto the CX board, the jumper block must be placed into the 65C816

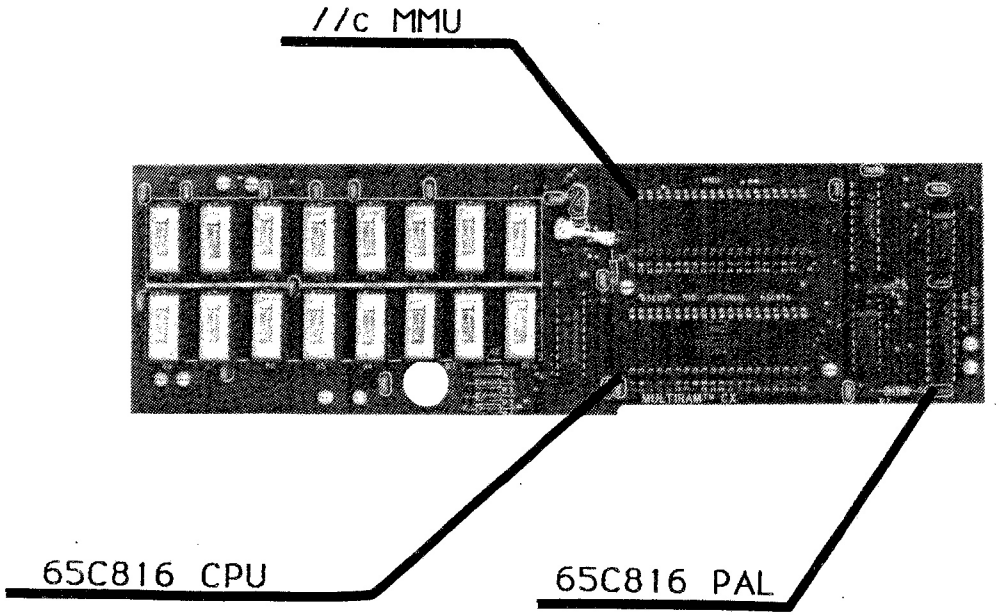


Figure 2.1 MultiRam CX Card

socket again in order for the CX card to work in the //c. Figure 2.2b shows that the jumper block must be inserted into the middle of the socket with two socket holes showing at the top and bottom of the socket.

With both the 65C816 and the 65C816 PAL inserted onto the MultiRam CX card, the card is almost ready to be installed into the //c. The MMU from the motherboard must also be inserted onto the CX card if you have not already done this. Refer to your MultiRam CX manual for further information on preparing the card for insertion into the //c.

Install the card in the //c as described in the MultiRam CX manual. Next, test the 65C816 option as described later in this chapter.

Inside Installation

If you have already installed the MultiRam CX card inside your //c, you will need to reopen the //c's case to install the 65C816 options.

Follow the instructions given in your MultiRam CX manual to reopen the case of the //c. Be sure to unplug the power cord from the //c before beginning its disassembly.

With the MultiRam CX card exposed and the keyboard out of the way, you may install the 65C816 options without removing the MultiRam CX card from the //c.

Use a spoon, as described in the MultiRam CX manual, or a flat blade to pry the 65C02 CPU out of the already installed card. The 65C02 is located in the socket at the bottom of the card identified as the "65C02 OR OPTIONAL 65C816" socket. Place the removed CPU into antistatic foam and safely store the 65C02 away as it will not be needed with the card.

Identify the 65C816 PAL socket. This socket is at the bottom right corner of the card labeled "OPTIONAL 16L8" and contains a jumper block. Pull off the jumber block with your fingers to expose the socket underneath. You will insert the 65C816 PAL into this socket. Store the jumper block away as you will no longer need it.

To insert both the 65C816 and the 65C816 PAL, follow the instructions in the preceeding section.

With the 65C816 and 65C816 PAL installed, you are now ready to test the 65C816 option. Do not reasemble the //c until you have tested the card for proper functioning.

TESTING THE 65C816 OPTION

To test the 65C816 for proper functioning, follow the testing procedures given in the MultiRam CX manual. As discussed in the manual, use the MultiRam utility disk from the MultiRam CX memory card when booting.

Figure 2.2a
Notch Position

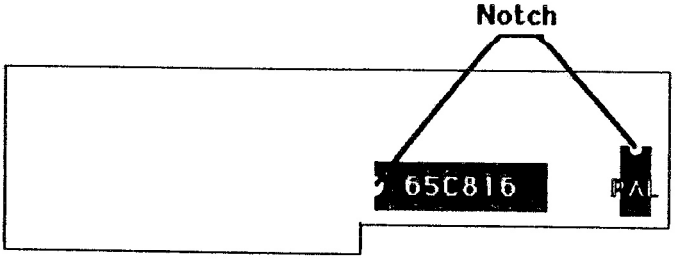
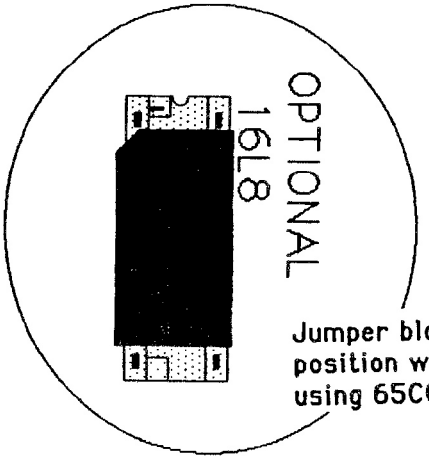


Figure 2.2b
Jumper Block Position



If the Utility disk successfully boots, run the RAM test diagnostic. If the RAM test works correctly as explained in the MultiRam CX manual, the 65C816 is working correctly.

If the Utility disk fails to boot and no display is seen or bars or broken lines fill the monitor screen, the 65C816 options may not be correctly installed.

First, check both of the 65C816 components, especially if you installed the 65C816 options with your card already installed in the //c and your card was working correctly before you inserted the 65C816 options.

- Does the notch of the 65C816 line up with the outline under it pointing to the left?
- Does the notch on the 65C816 PAL point to the top of the card?
- Did all pins of each component go into their respective socket hole or is a pin or pins bent out or bent under the component?

If none of the previous errors have occured, you may not have inserted the MMU onto the card correctly or the MultiRam CX card may not be installed into your //c correctly. Refer to the MultiRam CX manual for troubleshooting information for these potential problems.

If, after checking both the installation of the 65C816 components on the card and the installation of the card itself, your card still does not boot a disk or run software correctly, please call our Customer Service Department for help.

A set of specific 65C816 test routines will be included on the 65C816 disk you will receive at a later date.

Chapter 3

PROGRAMMING THE MULTIRAM 65C816

The Apple //c with the MultiRam CX Card and MultiRam 65C816 option installed becomes three computers in one:

1. A standard 8-bit, 128K, 65C02 computer with 64K of main memory and one 64K bank of bank-switched auxiliary memory.
2. An 8-bit, 640K, 65C02 computer with 64K of main memory and up to 9 banks of 64K bank-switched auxiliary memory.
3. A 16-bit, 65C816 computer with up to 640K of directly addressable memory.

The following sections provide general information about the 65C816 processor for assembly language programmers and offer some assembly language programming guidelines that should be used with the MultiRam 65C816. Chapter 4 further explains how the memory of the MultiRam enhanced Apple //c computer is addressed by the 65C816 processor,

65C816 GENERAL SPECIFICATIONS

The 65C816 CPU, used in the MultiRam CX kit, is a CMOS microprocessor with 16-bit internal architecture for arithmetic and logical operations and an 8-bit bidirectional data bus for exchanging information with external memory or peripherals. Advanced CMOS design means low power consumption for the 65C816.

The 65C816 features total software compatibility with the earlier 8-bit CMOS and NMOS 6502 microprocessors. A software switch, the Emulation select flag (E) in the 8-bit Processor Status Register of the 65C816, determines whether the 65C816 is in 65C02 8-bit "emulation" mode or 16-bit "native" mode (see Figure 3.1, the Programming Model). Emulation mode allows complete compatibility for existing Apple software and hardware.

The 65C816 features a 16-bit Accumulator, X and Y Index, and Stack registers. The shaded portion of the Programming Model shows that these registers revert to the same 8-bit status of the 6502 when the 65C816 is in emulation mode.

Two new Processor Status flags determine the size of the Accumulator and Index registers. The Memory Select (M) flag selects between an 8-bit and 16-bit Accumulator (8-bit = M flag set or 1; 16-bit = M flag clear or 0). The Index Select (X) flag selects between 8-bit and 16-bit X and Y Index registers (8-bit = X flag set or 1; 16-bit = X flag clear or 0). M and X are set to 1 in emulation mode.

A new 16-bit Direct Page register is available in the 65C816 which extends the 65C02's zero page addressing (256 bytes with the 65C02) to a 64K range in bank 0 (zero) of memory. Seperate Program Bank and Data Bank registers permit 24-bit memory addressing so that 16 megabytes of memory can be directly addressed. See Appendix B for more details.

The 65C816 offers 24 addressing modes - 13 original 6502 modes plus 11 new addressing modes with 91 instructions using 255 opcodes. New block move commands have been added to move large ranges of memory given the number of bytes to move and the starting addresses of the source and destination ranges. See Appendix I for more details.

Eight new opcodes to transfer values between registers and push and pull values from the stack allow the new Data Bank, Program Bank, and Direct Page registers to be directly controlled from within programs. A number of other new opcodes have been added in addition to these (see Appendices C and D).

Following is more detailed information about the more important features of the 65C816.

Selecting Operating Modes

When power is first applied to the 65C816 on the MultiRam //EX card, the 65C816 automatically enters 8-bit, 65C02 emulation mode. In emulation mode, the Accumulator, X and Y registers, and the Stack all are 8-bit.

Three flags on the Processor Status register are set to emulation mode at power up. These are the Emulation flag (E), the Memory Select flag (M), and the Index Select flag (X).

— The Emulation Flag & XCE —

The Emulation flag controls the operating state of the 65C816. When the flag is set (E=1), the 65C816 is in emulation mode. When the flag is clear (E=0), the 65C816 is in 16-bit native mode.

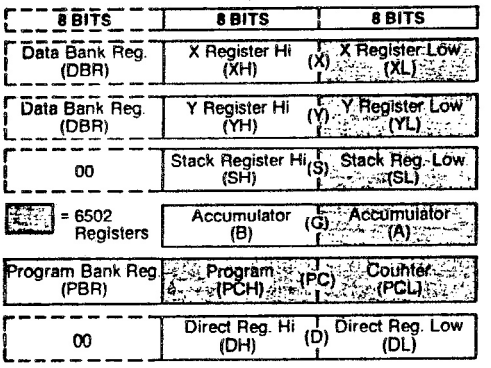
Notice that Figure 3.1 shows the E flag above the Carry flag. These flags are depicted this way as the Carry flag and a new opcode, XCE, determine the state of the emulation bit.

When the instruction XCE is encountered in a program, the Carry flag bit, clear or set, is exchanged with the Emulation flag — e(X)change (C)arry bit with (E)mulation bit. So, the mode of the 65C816 can be selected by using one of these two groups of instructions:

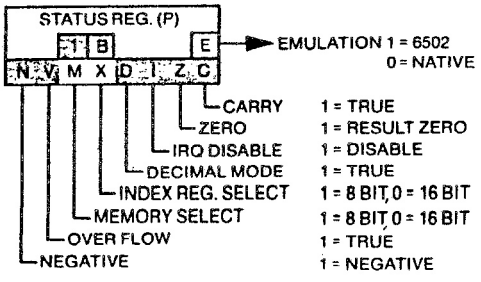
Select 16-bit Native Mode		Select 8-bit Emulation Mode	
CLC	Clear Carry bit	SEC	Set Carry bit
XCE	(E = 0)	XCE	(E = 1)

Figure 3.1

W65C816 Processor Programming Model



Status Register Coding



If the state of the Carry flag prior to the exchange is required for later use, the Processor Status should be pushed onto the stack or otherwise saved for later use before the XCE instruction.

After the emulation bit is cleared for 16-bit native operation, the processor can be brought back to emulation mode. The Accumulator, X and Y registers revert to 8-bit when the emulation mode is turned on; they must be returned to 16-bit each time a program crosses from emulation to native mode, if 16-bit operation is desired.

A reset will cause emulation mode to be invoked. Programs must account for a reset and provide the code necessary to return to 16-bit mode, if that is what is desired, through user defined reset vectors.

Many programs written specifically for the 65C816 will probably enter 16-bit mode and not return to emulation mode. This is feasible because in 16-bit mode, both 8-bit and 16-bit register operation is possible (explained in the next section); you do not have to reenter emulation mode to use 8-bit values.

INTERRUPTS

Appendix H shows differences between emulation and native modes on the 65C816 in a number of areas and contrasts both modes to the 65C02 and 6502.

One major difference between emulation and native mode is the assigned memory location of interrupt vectors. If your program is to handle Interrupt Request (IRQ) by a peripheral device, you would need to supply vectors and interrupt handler routines in the appropriate RAM locations.

Vector Locations	65C02 Emulation	65C816
ABORT	$00 FFF8,9	$00 FFE8,9
BRK	00 FFFE,F	00 FFE6,7
COP	00 FFF4,5	00 FFE4,5
IRQ	00 FFFE,F	00 FFEE,F
NMI	00 FFFA,B	00 FFEA,B
RES	00 FFFC,D	00 FFFC,D

You should enter emulation mode if you are using firmware routines from the Apple or using a language, operating system or peripheral device that is dependent upon interrupt locations from firmware (see "Interrupts" in Chapter 4 for more details).

CPU IDENTIFICATION

You may want to shift back and forth between emulation and native mode if you need to write a program that can run on a 6502 or 65C02 but take advantage of the 65C816's opcodes. Such a program should be modular.

Write those segments of your program that do not benefit from 16-bit operation in 6502 or 65C02 code. Write routines that would benefit from 16-bit operation in both 16-bit mode and 8-bit mode and use the 8-bit module when a 65C02 Apple is running the program and a 16-bit module when a 65C816 Apple is running the same program.

To design such a modular program, you would need to identify the processor running the program. Because differences exist in types and numbers of opcodes between the 6502 used in Apple II's and //e's, and the 65C02 used in enhanced //e's and //c's, your routine should ideally identify a 6502, 65C02, 65C802 (which cannot directly address more than 64K but has the same opcodes as a 65C816), and a 65C816.

A sample processor identification routine is:

LDA #$FF	Prepare to test the zero flag.
INC	INC ($1A) will increment accumulator to zero for 65C02 & 65C802/816 and set the zero flag; no operation occurs in the 6502.
BNE 6502	Zero flag clear, therefore no increment of A register and a 6502 was identified.
REP #$02	REP ($C2) resets Status register bits only in a 65C802/816 but performs no operation in the 65C02; REP #$02 resets the zero flag from set (1), where the INC instruction should have left it, to clear (0).
BEQ 65C02	Zero flag still set, therefore REP command did not occur and a 65C02 was identified
	65C802 or 65C816 identified; further test must be made based on long addressing; 65C802 does not directly address more than 64K directly; test not conclusive if only 64K available.

Note in the example that the REP command, an opcode that is not available on the 65C02, does work even when the 65C816 is in emulation mode. All 65C816 opcodes which do not require 16-bit registers to properly operate will work in emulation mode.

— Memory/Index Flags & SEP and REP —

The Accumulator, X, and Y registers are all set to 8-bit when the 16-bit native mode is first entered. These registers will also revert to 8-bit width if, after they have been put into 16-bit width, the emulation flag is again turned on through program control or a reset.

Separate Status register bits allow you to control the state of the Accumulator, X, and Y registers. These are the Memory (M) and Index (X) Select flags. As Figure 3.1 shows, when these flags are set, 8-bit A, X and Y registers result; when cleared, 16-bit registers result. The two flags may be set or cleared independently of one another.

Rather than use older, more code intensive methods to reset the flags of the status register, such as pushing a byte onto the stack and then pulling it off the stack with a PLP, two new shorter opcodes, REP and SEP, are most commonly used to set the M and X bits as well as other bits of the Status register.

The instruction REP followed by a byte, will reset the bits of the Processor Status register. Any bits of the Status register that are set and match the set bits of the byte following REP will be cleared. Any clear bit in the Status register is unaffected. SEP does the opposite and sets a cleared Status bit if the corresponding bit of the byte following SEP is set.

Here are some examples:

```
          NVMXDIZC  Processor Status Flags
          --------
          xx11xxxx  Status bits before REP
                    M & X bits set; A, X & Y reg. = 8-bit

REP #$30 %00110000  use a bit pattern to clear only M & X Status bits

          xx00xxxx  Status bits after REP
                    M & X bits clear; A, X & Y reg. = 16-bit

SEP #$20 %00100000  use a bit pattern to set the M bit

          xx10xxxx  Status bits after SEP
                    M set so A reg. = 8-bit; X clear so X & Y = 16-bit

                    x = not affected
```

Most often, after the XCE instruction is given for 16-bit native mode, you would next select the width of the registers you wish to use for the main program using SEP or REP commands.

Most assemblers for the 65C816 have a psedo-opcode that must be used when switching register size in order for the assembler to keep track of register width. You may wish to construct a macro that will not only automatically issue the correct REP or SEP command (or PHP or PLP) but will also give the assembler the pseudo-opcode it requires for proper assembly.

Note from the Programming Model figure that the Accumulator is named C when in 16-bit mode. When in 8-bit mode, the Accumulator has two sections, A and B. As the Model and Appendix A indicate, the upper 8-bits (B) may be used for temporary storage with the XBA instruction when in 8-bit mode.

Register Work

Because the 65C816 allows you to control the size of the most commonly used registers when the processor is in native mode, you need to keep track of the existing register size when programming; otherwise undesired results can occur.

For example, if you are using a 16-bit Accumulator, any load instruction will gather two consecutive bytes, not one as would be the case with an 8-bit Accumulator. A LDA $1000 in 16-bit mode would first load into the Accumulator the value stored at $1000 and then load the value stored at $1001. In 8-bit mode, only the value at $1000 would have been loaded into the Accumulator. If you store a one byte 8-bit value with an 8-bit Accumulator and then load the value back into a 16-bit Accumulator, you will not load the desired value. You have created an error that could crash your program. The same result would hold true with the X and Y registers.

If you are using the X or Y registers for indexed addressing, similar care should be taken. For example, if you have a table of values that spans more than 256 bytes and you are using X or Y registers to look up a specific table value from a location more than 256 bytes into the table, you will not get the expected value if the X or Y register is 8-bits wide; you will instead get an incorrect value from within the first 256 bytes.

Stack manipulation could become a source for multiple program crashes if you are not aware of the size of the value that was pushed onto the stack when the time comes to pull off that value.

For example, if you push a 16-bit, two byte value onto the stack and are using an 8-bit Accumulator when you try to pull the value off the stack into the Accumulator, the resulting byte in the Accumulator will not be the original value. Further, the stack will now contain an unmatched, invalid byte. You have created two potential sources for error with this mistake: an incorrect Accumulator value and an unbalanced stack that, among other potential errors, would generate an incorrect address for the next RTS.

One method to avoid errors related to register size is to declare a standard or global M and X register condition as part of your initialization routine. Register size would then shift only at the start of subroutines with REP or SEP commands and M and X would be restored upon exit of subroutines to the standard M and X setting with REP or SEP commands. A very useful standard configuration would be an 8-bit Accumulator and 16-bit X and Y registers.

Another method of restoring M and X in subroutines would be to push the Processor Status register (PHP) at entry and pull it (PLP) at subroutine exit. Subroutines using register sizes differing from the standard should not call other subroutines using still different size registers without resorting to the PHP, PLP sequence.

In native mode, routines that service interrupt requests do not need to push the Processor Status register upon entry as an IRQ in the 65C816 pushes the Program Bank register, Program Counter, and Processor Status registers onto the stack and the RTI command restores these registers in reverse order (see Appendix G).

If you are shifting between emulation mode and native mode within your program, each time you reenter 16-bit native code you should restore the standard M and X settings. The M and X bits are always set to 1 when emulation mode is entered.

Direct Page & Stack Addressing

The 65C816 allows programmer control over both the starting locations of zero page and the stack anywhere within the first 64K of memory, bank 0 (zero).

— The Direct Page —

The 65C816 has a Direct Page register which the 65C02 and 6502 do not have. The Direct Page register is a 16-bit register that functions only when the 65C816 is in native mode.

The Direct Page register replaces zero page addressing with "direct page addressing" permitting zero page addressing to take place anywhere in the first 64K of memory. When a zero page address is referenced, the value in the Direct Bank register is added to the byte following the zero page opcode to form the effective direct page address.

The 6502 and 65C02, in contrast, do zero page addressing in only the first 256 bytes of memory. Therefore, direct page addressing extends the advantages of zero page addressing to 64K of memory: zero page addressing takes less clock cycles to perform than absolute addressing and involves less bytes.

You can set the start of zero page locations to be anywhere in the first 64K of memory by placing the starting value of the desired zero page into the Direct Page register. The register is set using a PLD opcode. The 65C816 must be in native mode (E=0) and a 16-bit value must be passed to the Direct Page register (M=0). Any reference to a zero page location adds the byte following the operand to the Direct Page address.

For example, the following code would set the starting zero page to $00 1000:

```
PEA $#1000      Push the 16-bit value $1000 onto the Stack. (See Appendix D for
                a summary of new push/pull commmands.)
PLD             Pull the $1000 off the Stack into the Direct Page regs.

- or -

LDA $#1000      Load the 16-bit value $1000 into the Accumulator C.
TCD             Transfer Accumulator C to the Direct Page register. (See
                Appendix D for a summary of new transfer commands.)
```

Now, if you use a zero page instruction such as LDA $80, the $80 is added to the contents of the Direct Page register to form the effective direct page address of $1080.

To maintain fastest direct page addressing speed, set the starting location of the direct page on a page boundary. If you set the low byte of the Direct Page to a non-zero location, you will add one cycle to processing time (i.e., a $1000 Direct Page register base address is faster than a $1001 base address).

If you set the Direct Page registers to $0000, then zero page addressing will occur as it would in the 6502 or 65C02.

If you switch back to emulation mode, the direct page addresses will still be in effect for zero page locations. Therefore, to maintain true compatibility, reset the Direct Page register to $0000 before returning to emulation mode. If you wish to return to 16-bit native mode, you should store the Direct Page in effect before returning to emulation.

Direct Page is set to $0000 upon power up or when a reset occurs.

— The Relocatable Stack —

The Stack Pointer register is a 16-bit value when the 65C816 is in native mode (E=0). Therefore, the stack may start anywhere within the first 64K of memory and may run as long as 64K.

To set the stack to a starting location, load the starting value you desire into a 16-bit Accumulator or X register and then transfer the register to the Stack register with a TCS (Transfer the 16-bit Accumulator) or TXS command.

If you return to emulation mode or if a reset occurs, the high byte of the Stack is automatically set to $01. If you will switch between native and emulation modes, you will then need to keep track of two stacks if you are using a 16-bit stack in a range other than $100-$1FF, the fixed stack location for the 6502 and 65C02.

Unlike the 6502 or 65C02, the stack in the 65C816 emulation mode does not roll over at a page boundary but simply continues building downward as far as the first 64K of memory allows. Therefore, in both emulation and native modes you should incorporate an overflow check to be certain you will not destroy memory locations below the desired end of the stack.

Memory Addressing Modes

The 65C816 permits memory beyond 64K to be addressed in two different ways, adds many new addressing modes to the 65C02 set, and adds a new block move command.

— Data & Program Bank Registers —

The 65C816 allows two methods to generate 24-bit addresses, a 24-bit address being able to access 16 megabytes. Two methods are allowed as a result of the Data Bank register being independent of the Program Bank register.

The Data Bank and Program Bank registers are two seperate 8-bit registers. The Data Bank register determines which bank of 64K will be used for memory transfers such as when using LDA, LDX, LDY, STA, STX, etc. The Program Bank register holds the bank address for all other instruction fetches.

SHORT ADDRESSING

The first method, the short addressing method, uses the Data Bank register and Program Bank register to establish segmented memory addresses. Since these registers are independent of one another, data may be used from a bank other than the bank where a program is running using two byte 16-bit addresses after a memory transfer opcode.

The following example shows one method for setting the Data Bank Register seperately from the Program Bank register:

```
LDA $#2     Select the 64K bank where data should come from (8-bit Acc.)
PHA         Push the bank number on the Stack.
PLB         Pull the data bank number into the Data Bank register.
```

Using short addressing, each bank register provides the first 8-bits of a 24-bit address and the bytes following a memory transfer instruction provide the remaining 16-bits of the address. Therefore, if the program that issues the preceeding instructions is running on the first 64K bank of memory, a LDA $1000 instruction would load data from address $02 1000 and not $00 1000.

This method of handling large memory is similar to the 8088/8086 but differs in that the Data Bank register does not rollover at a 64K boundary but increments to the next higher location providing for linear, unbroken data ranges.

If the Data Bank and Program Bank registers point to different banks and you desire to reset the Data Bank to the Program Bank value, a quick way to do so is to push the Program Bank register on the Stack with a PHK command and then pull that bank into the Data Register with a PLB command.

Both the bank registers are initialized to zero on processor start or during a reset. The bank registers will not change back to bank zero if you go back to emulation mode from native mode; so a return to emulation mode should take place on bank 0 (zero). More on this subject in Chapter 4 when MultiRam //c memory is discussed.

LONG ADDRESSING

The second addressing method, long addressing, uses linear addressing with three byte addresses. This is the simplest and least confusing of the two methods.

For example, given the same example as before, if your program is running in bank 0 and you want to load some data stored in bank 2, you would use a long address command like LDAL $02 1000 where the second, third, and forth byte determine the 24-bit address. Each assembler uses a different pseudo-opcode for long addressing so the LDAL may not be the code your assembler will use.

Appendix I shows that the only instructions affecting the Program Bank register value are JML, JMP Absolute Long, JSL, RTI, and RTL.

Note that the current 65C816 does not automatically increment the Program Bank register when a program crosses a bank range. The register rolls over. You must jump to the new bank where the program is to continue if your routine is approaching the end of a bank. Future versions of the 65C816 may not support rollover so you should not build a rollover feature into your programs.

— Addressing Modes —

The 65C816 has twenty-four address modes for long addressing. These modes are detailed in Appendix I.

Two addressing modes not offered with the 6502 or 65C02 are the Stack Relative Addressing Modes and the Block Move modes.

The Stack Relative modes use the stack pointer and an address following the stack relative instruction to form an effective address. The mode is available in both plain and indirect indexed modes. The stack may be used to easily pass data or data addresses to subroutines with these instructions, especially when used along with new stack push (PEA, PEI, PER) and transfer (TCS and TSC) opcodes.

This addressing mode would be very helpful for higher level language, like Pascal, in passing parameters between program segments. A calling routine can push data addresses or data on the stack to be passed to another routine, the called routine can then pull the values from the stack using simple stack relative commands, manipulate the values, and then push the resultant values back so that the changed values or addresses can be easily recovered by the calling routine.

Block move instructions MVP (Block Move Negative) and MVN (Block Move Positive) automatically copy a block of memory (up to 64K) given only the starting source and destination addresses and number of bytes to move. The X register holds the beginning 16-bit address of the first source byte to move; the Y register holds the beginning 16-bit address of the first destination byte; and the 16-bit Accumulator holds the number of bytes to move less one. The first byte following the move instruction is the high byte of the 24-bit destination address; the second is the high byte of the 24-bit source address.

The MVP instruction increments X and Y and decrements the Accumulator while MVN decrements all registers. The move ends when the Accumulator reaches zero. The Accumulator and X and Y registers must be set to 16-bit (M & X Processor Status bits = 0).

The following example will copy the data in $1000-$4FFF from bank 0 (zero) into $2000-$5FFF on bank 1:

```
LDA #$3FFF      Move the 16K bytes
LDX #$1000      starting at location $1000
LDY #$2000      to location $2000
MVP 1,0         with the source on bank 0 and the destination on bank 1 and fill
                the 16K destination range from $2000 up.
```

Block moves require far less time and much, much less code to implement than would comparable 8-bit move routines as indexes are automatically incremented and decremented by the processor.

Block moves reset the Data Bank register to the destination bank. Therefore, if you wish to use a different data bank when you exit a move routine, you will have to place a new value into the Data bank register using a push and a PLB command or a TCD transfer.

Chapter 4

MULTIRAM MEMORY ORGANIZATION

The 65C816 microprocessor has a 16 megabyte address range with its 24-bit address bus. The MultiRam CX card allows the 65C816 to directly address slightly over 8 megabytes of memory with future expansion possible. The CX card with 65C816 option can therefore directly address the 640K of memory the MultiRam CX memory card creates in the //c.

Figure 4.1 shows the standard 128K //c memory map. Figure 4.2 shows the memory map with maximum MultiRam memory and the CX card in 8-bit emulation mode. Figure 4.3 shows the memory map with maximum MultiRam memory and the CX card in 16-bit native mode.

Note that the label memory "bank" is used in all of the figures. The word has a different meaning in each figure.

A "bank" of memory in the standard //c refers to one of two areas of 64K RAM: main or auxiliary memory. Since the 6502 or 65C02 used in the Apple //c cannot address more than 64K at a time, the banks of memory are not contiguous and so are not numbered. The 65C02 or 65C816 in emulation mode can draw its 64K addresses in several different ways (more on standard addressing later) from these unrelated RAM banks.

With MultiRam memory added to the //c and the CX's optional 65C816 in emulation mode, the word "bank" can refer to the MultiRam 64K RAM banks shown in Figure 4.2. Note that these banks are numbered.

MultiRam banks are numbered for programmers convenience only as the numbers have no real meaning to the 65C02 or the 65C816 in emulation mode because they cannot directly address more than 64K of memory. The numbers do allow programmers to select which 64K bank of MultiRam memory is brought on line as standard //c auxiliary memory using a soft switch described later. The //c's standard 64K of auxiliary memory is numbered bank 0 (zero) with following MultiRam 64K banks numbered in ascending sequence.

MultiRam 64K memory bank numbering <u>is</u> significant to the 65C816 because the 65C816 can address up to 16 megabytes of memory. A 24-bit address is required to address 16 megabytes. The high byte of the 65C816's 24-bit address, the XX in a "$XX xxxx" 24-bit address, represents a multiple of 64K and is what is refered to by the 65C816 in native mode as a "bank."

MultiRam memory cards were designed from the beginning to support the 65C816's ability to linearly address 16-megabytes; therefore, the 64K banks are numbered logically and sequentially; i.e., bank 1, bank 2, bank 3, etc. Contiguous, linearly mapped memory means that the 65C816 can address data across MultiRam's 64K boundaries in the Apple without having to resort to data segmenting (although that is possible as was described in the previous section on programming the 65C816).

In the Apple, the CX's 65C816 addresses motherboard memory as bank 0 (zero) or $00 xxxx, MultiRam bank 1 as bank $01 xxxx, etc. Because there can be only one bank 0 for the 65C816, the standard auxiliary memory of the //c which contains special video functions and is numbered as bank 0 (zero) for the 65C02 or 65C816 in emulation mode, is remapped into a different bank number, bank $8F xxxx. This remapping potentially allows 8 megabytes of directly addressable, linear memory and easy addressing of the special area of auxiliary memory (more on this subject later). The same is true of the 65C816 card for the Apple //e.

Locations within the 8 megabyte range now addressable in the Apple by the MultiRam 65C816 can be divided into three distinct areas: I/O-ROM, standard //e 128K memory, and MultiRam memory. The MultiRam 65C816 addresses these three areas in the //c differently depending upon whether the 65C816 is in 65C02 8-bit emulation mode or 16-bit native mode.

Following are explanations of these differences. The explanation assumes some familiarity with normal Apple memory addressing and assumes you have read Chapter 3.

I/O-ROM Addresses

The area on the //c motherboard identified as hardware addresses in Figure 4.1 is divided into three address ranges: Built-in I/O switches ($C000-$C0FF), Port I/O ($C100-$CFFF) and Applesoft/Monitor ROM ($D000-$FFFF).

— Built-In I/O Switches —

A program can read and write to I/O locations in the range $C000-$C0FF to control internal and external devices through electronic switches connected to the Apple's processor — keyboard, video, speaker, joysticks, etc. — as well as determining which area of main or auxiliary memory is brought into the address range of the 65C02 CPU.

When the 65C816 is in 8-bit emulation mode, these hardware locations are addressed as the 65C02 would normally address them. For example, a LDA $C000 would check the keyboard strobe for input.

These I/O locations are unaffected when the 65C816 enters 16-bit native mode and remain mapped into only bank zero at $00 C000 to $00 C0FF when the 65C816 is in 16-bit native mode. No other area of memory on other banks corresonds to these addresses. For example, a location of $02 C000 is actual RAM memory and addressing it does not affect the keyboard strobe.

To address the built-in I/O switches in long addressing mode, simply set the bank number to zero and set the lower two bytes of the three byte long address to the built-in I/O location. For example, to check the keyboard strobe for input, use a LDAL $00 C000 command.

Figure 4.1
Standard //c Memory

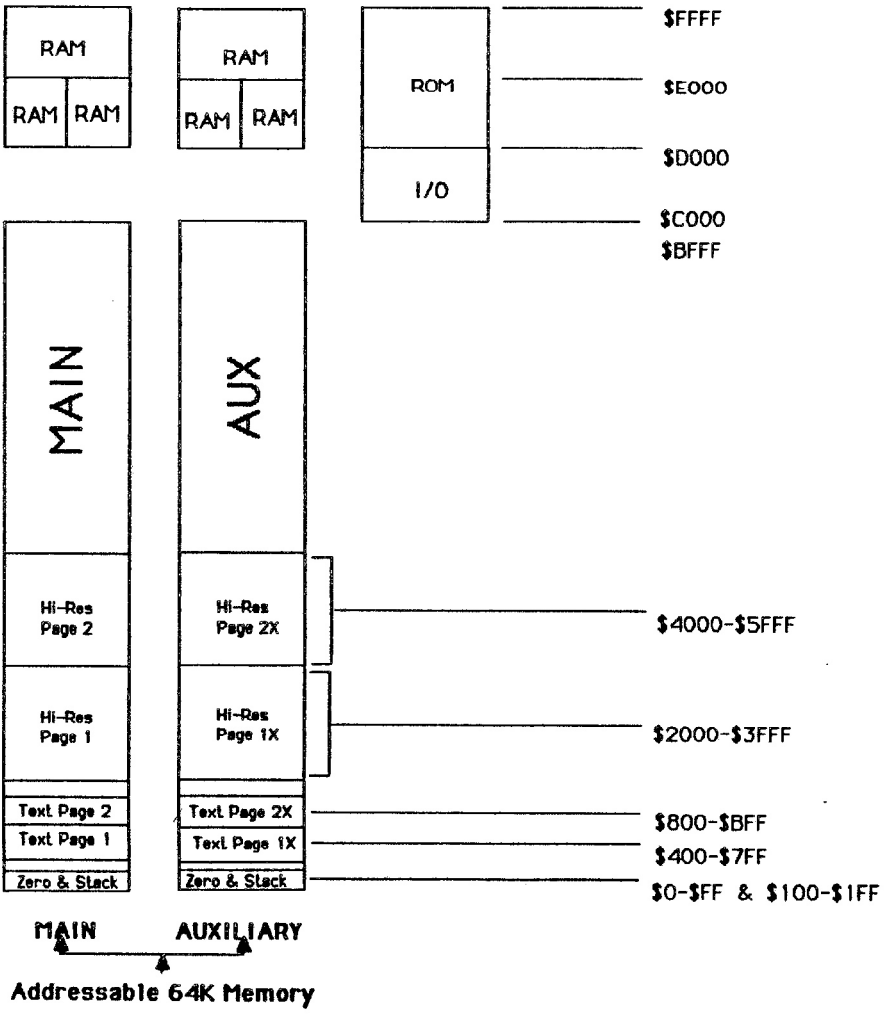


Figure 4.2

MultiRam CX Memory Map (EmulationMode)

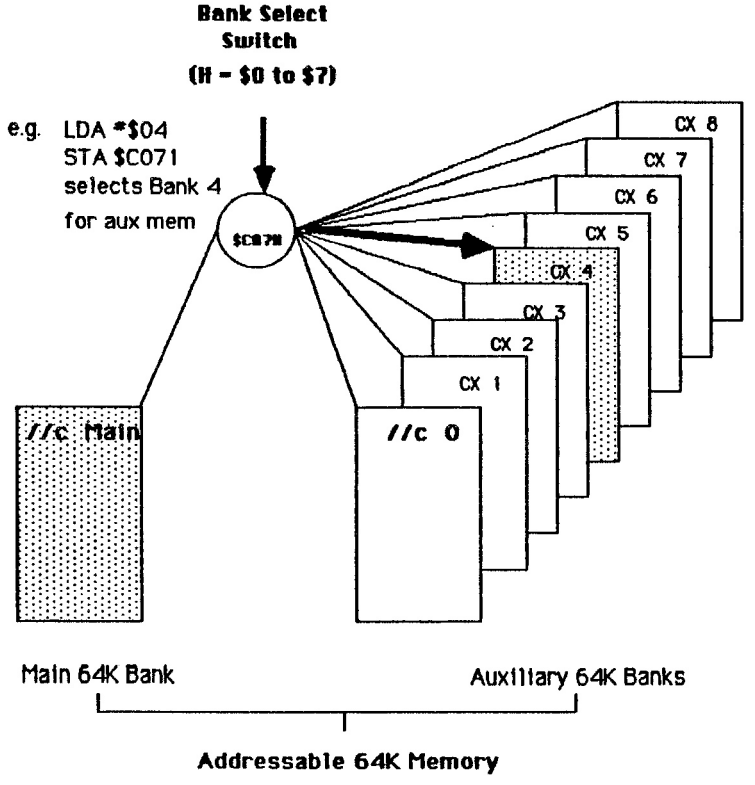
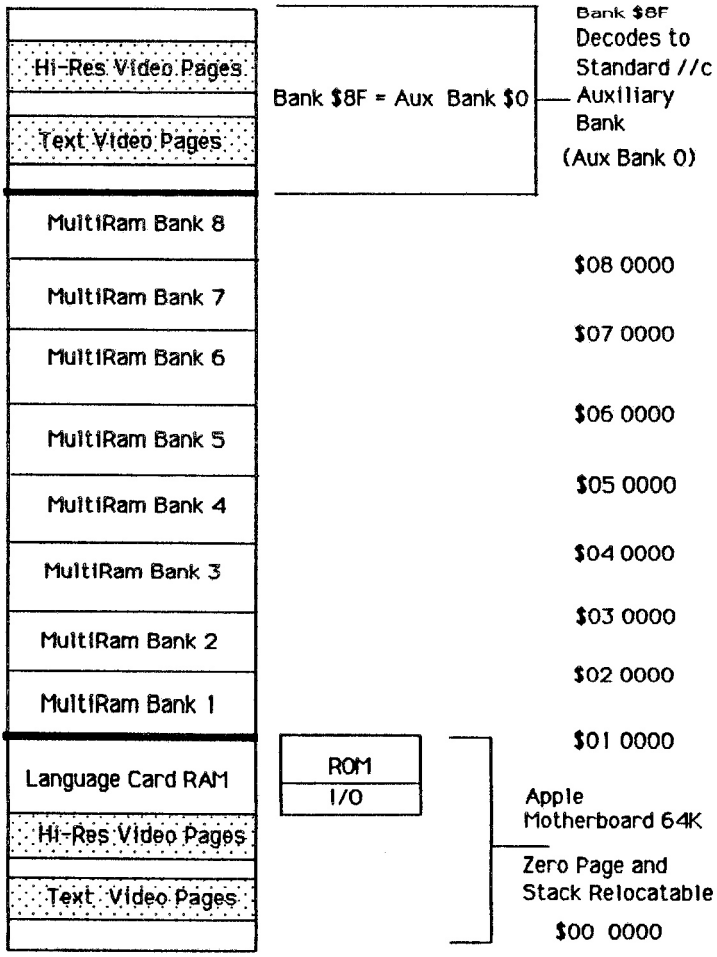


Figure 4.3

16-bit Memory Map



If you address the built-in I/O switches in short address mode, you must be certain to have the Data Bank Register set to bank 0 (zero) to reach the I/O switches. If the Data Bank register has been set to any 64K bank other than bank 0 (zero), you would be reading or writing to a RAM location rather than an I/O soft switch. For example, the same instruction LDA $C000 would address the keyboard strobe if the Data Bank register is $00 but would address RAM location $02 C000 if the Data Bank register was $02.

DO NOT reset the Direct Page register or the stack pointer into the $00 C000 - $00 COFF range. This area of memory is not RAM and trying to use it for direct page or stack locations will not work.

— Port I/O —

The address range $C100-$CFFF is used by ROMS in the //c to control peripheral devices connected to the //c's ports, devices such as printers, modems, a mouse, etc. Firmware routines in //c ROM in this range are addressed normally when the 65C816 is in 8-bit 65C02 emulation mode. These firmware locations, when the 65C816 is in 16-bit native mode, remain mapped into bank 0 (zero) from $00 C100 to $00 CFFF and are not mapped into any other memory addresses.

When in 16-bit mode, because the firmware on ROM in the //c is written in 8-bit 65C02 code, you should not attempt to pass Accumulator or X and Y register values to firmware unless the Accumulator and X and Y registers are set to 8-bit width using the Memory Select bit M and the Index Select Bit X.

If you believe there is any chance that code in the //c's ROM could be falsely decoded by the 65C816, you should set the 65C816 to emulation mode when addressing peripheral devices.

Also, because interrupts are generated by the //c's hardware for many functions (mouse movement, vertical blanking, keyboard buffering, etc.), you may wish to set the 65C816 to 65C02 emulation mode because the 65C816 interrupt vectors, when the 65C816 is in native mode, are in different locations than the standard 65C02 (discussed in the next section). If you do address peripheral's in 16-bit mode, then you must turn off motherboard ROMs and use the RAM in the Language Card area of motherboard memory to set up your own interrupt vectors. See the next section for more information.

Both the Data Bank register and Program Bank register should be set to bank 0 (zero) if you access the firmware routines in a //c in 16-bit mode. If you use a long JMP (JML) or a long JSR (JSL) to access peripheral routines, you should have no problems with the Program Bank register. Be sure to set the Data Bank register to zero, however, as the firmware routines do use scratchpad locations in motherboard RAM to store pertinent operating information.

Do not attempt to use peripheral ROM locations for direct page addressing and do not try to set the stack in this area of memory.

— Applesoft/Monitor ROMS —

Applesoft and monitor ROM routines operate as usual when the 65C816 is in emulation mode.

Not all Applesoft routines may operate properly when the 65C816 is in native mode. If you intend to use Applesoft programs to call 65C816 routines, you should keep the 65C816 in emulation mode when dealing with the Applesoft program and switch it into native mode only when using an assembly language subroutine. Other Basic languages will soon be available for the Apple that are written in 65C816 code (see Chapter 5).

Most, but not all, monitor routines should work correctly when the 65C816 is in native mode. One set of routines which will not work correctly are those dealing with interrupt requests (IRQ).

INTERRUPTS

As Table 4.1 shows, interrupt vectors differ when the 65C816 is in emulation versus native mode. The most critical of these are the IRQ (Interrupt Request), Reset, and NMI (Non-Maskable) interrupts.

These three interrupt vectors are frozen into the Apple monitor's firmware at the locations shown in the table for 65C816 emulation mode. Both the IRQ and NMI vectors exist at different locations in emulation mode than when in native mode. Therefore, if you cannot switch out the motherboard ROM for a valid reason, you could put the 65C816 into emulation mode whenever you expect an IRQ may be issued or mask out interrupts with a SEI opcode.

If you can switch motherboard ROM out of the memory map and switch the Language Card RAM memory into addressable memory through use of the Language Card soft switches, you could set your own 16-bit interrupt handlers at the 65C816 16-bit vector locations in RAM and still use peripheral devices in 16-bit mode. Appendix G, address modes 21a and 21c, show the registers pushed on the stack during an interrupt and shows the order they are restored.

TABLE 4.1 Interrupt Vector Locations

Interrupt Vectors	65C02 Emulation	65C816
ABORT	$00 FFF8,9	$00 FFE8,9
BRK	00 FFFE,F	00 FFE6,7
COP	00 FFF4,5	00 FFE4,5
IRQ	00 FFFE,F	00 FFEE,F
NMI	00 FFFA,B	00 FFEA,B
RES	00 FFFC,D	00 FFFC,D

Note that the Reset vector is shown at the same locations in both modes. This is because the 65C816 reverts back to emulation mode if a Reset occurs during native mode as Table 4.2 shows:

TABLE 4.2 Register Conditioning After Reset

Direct Page = $0000					Stack High Byte = $01				
Data Bank = $00					X Index High Byte = $00				
Program Bank = $00					Y Index High Byte = $00				
	N	V	M	X	D	I	Z	C/E	
Processor Status =	*	*	1	1	0	1	*	*/1	

* = not initialized

If you wish to have your program restored to 16-bit native mode after a Reset, you must preset the standard Apple $3F2, $3F3, and $3F4 soft entry vectors and power-up byte to point to your restoration code.

As Appendix G, address modes 21a and 21c show, the Program Bank register, Program Counter, and Processor are all pushed on the stack during a Reset; the Data Bank, Direct Page, and high byte of the Stack registers are not. You may want to store the Data Bank, Direct Page, and Stack High byte registers to locations unaffected by a Reset in order to return them to their pre-interrupt status. You may also want to save the X, Y, and Accumulator register values in your own interrupt handler routines to restore them after a Reset from the keyboard occurs while in 16-bit mode.

If your interrupt handler code is in 16-bit mode, you should be careful of the size used by the Accumulator and Index registers. The Processor Status register pushed on the stack contains the settings of M and X at the interrupt point. If you change these within a 16-bit interrupt handling routine, be sure to restore them at exit if you do stack manipulation which could disturb the pushed value of the Processor Status register.

Standard //c Addresses

The standard 128K memory of the //c consists of two independent 64K RAM segments: main memory and auxiliary memory. Figure 4.1 shows the normal memory map of the 128K Apple //c.

The standard 65C02 in the Apple //c can directly address only 64K of memory at one time in the hexadecimal range $0000 to $FFFF because the 65C02 only has a 16-bit wide address bus. The 65C816 when in 8-bit emulation mode has this same 64K address limitation.

Because the standard 65C02 can address only 64K at a time, software controlled latches or memory soft switches exist to select which portions of the two RAM areas are addressed by the 65C02 at any given time. The 64K recognized by the CPU can consist of all motherboard memory, all auxiliary memory, or combinations of both main and auxiliary memory.

Two sets of software switches determine which combination of main and auxiliary memory totalling 64K are addressed by the CPU. These are the auxiliary switches and video display switches identified in Table 4.3.

Figure 4.4 shows one common example of switching where the 65C02 reads and writes to main memory in the address range $0-$1FF and $D000-$FFFF and reads and writes to auxiliary memory in the range $200-$BFFF. The dark shading shows these areas in the two different RAM banks.

Both main and auxiliary memory are further divided into 48K main memory and 16K Language Card areas. The Language Card memory areas share address locations with the //c's Applesoft/Monitor ROM addresses. RAM or firmware ROM addresses are soft-switch selected into the //c's memory map as are main and auxiliary locations with the switches shown in Table 4.3.

When the 65C816 is in 8-bit emulation mode, these three sets of memory switches must be used to address the auxiliary memory area. Other 64K banks of MultiRam's memory are switched into and out of the auxiliary memory area as is described later.

When the 65C816 is in 16-bit native mode, a number of these three sets of switches are no longer needed to address memory as the 65C816 can directly address 16 megabytes of memory. The same caution about having bank 0 selected as the Data Bank register before addressing these switches, as was described earlier for all switches in the I/O area, does still apply.

— Language Card Memory—

Language card switches determine whether ROM or RAM is to be addressed by the 65C02 when reading from and writing to locations in the $D000-$FFFF range, either in main or auxiliary memory . ROM addresses in this range contain Applesoft and the //c system monitor. These switches and their actions are shown in Table 4.3.

When the 65C816 is in emulation mode, the Language Card soft switches for both main and auxiliary meory must be used to access the 16K area they control.

When the 65C816 is in native mode, only the motherboard switches must be used to address the Language Card area of main memory. Auxiliary memory Language Card RAM is treated as directly addressable memory and is addresssed like the rest of auxiliary memory described in the next section.

Therefore, the motherboard Language Card area is the <u>only</u> memory in the Apple that must still be soft switch selected.

TABLE 4.3 //c Memory Management Soft Switches

Switch Name	Hex Location	Function		
LANGUAGE CARD				
4K RAM Bank 2	$C080	Read: RAM		
	$C081	Read: ROM	Write: RAM	
	$C082	Read: ROM		
	$C083	Read: RAM	Write: RAM	*
	$C084	Read: RAM		
	$C085	Read: ROM	Write: RAM	
	$C086	Read: ROM		
	$C087	Read: RAM	Write: RAM	*
4K RAM Bank 1	$C088	Read: RAM		
	$C089	Read: ROM	Write: RAM	
	$C08A	Read: ROM		
	$C08B	Read: RAM	Write: RAM	*
	$C08C	Read: RAM		
	$C08D	Read: ROM	Write: RAM	
	$C08E	Read: ROM		
	$C08F	Read: RAM	Write: RAM	*
		* Requires 2 successive reads to enable		
AUXILIARY BANK				
RAMRD	$C013	Read RAMRD switch		
	$C002	OFF: Read Main 48K ($0200-BFFF)		
	$C003	ON: Read Aux. 48K		
RAMWRT	$C014	Read RAMWRT switch		
	$C004	OFF: Write to Main 48K		
	$C005	ON: Write to Aux. 48K		
ALTZP	$C016	Read ALTZP switch		
	$C008	OFF: Use Main memory, Page 0, Stack bank-switched (16K) memory		
	$C009	ON: Use Aux. memory, Page 0, stack bank-switched memory		

TABLE 4.3 //c Memory Management Soft Switches

Switch Name	Hex Location	Function
VIDEO DISPLAY		
80COL	$C01F	Read 80COL switch
	$C00C	OFF: Display 40-columns
	$C00D	ON: Display 80-columns
80STORE	$C018	Read 80STORE switch
	$C000	OFF: Enable RAMRD, RAMWRT switches for video pages
	$C001	ON: Override RAMRD, RAMWRT switches for video pages (use PAGE2 switch)
PAGE2	$C01C	Read PAGE2 switch
	$C054	OFF: 1. Display video page 1 using RAMRD, RAMWRT (80STORE off)
		2. Display video page 1 using main memory (80STORE on)
	$C055	ON: 1. Display video page 2 using RAMRD, RAMWRT (80STORE off)
		2. Display video page 1 using AUX. memory (80STORE on)
TEXT	$C01A	Read TEXT switch
	$C050	OFF: Display graphics
	$C051	ON: Display text
MIXED	$C01B	Read MIXED switch
	$C052	OFF: Full graphics
	$C053	ON: Mixed text and graphics
HIRES	$C01D	Read HIRES switch
	$C056	OFF: Low-res graphics selected
	$C057	ON: Hi-res graphics selected
DHIRES	$C07F	Read : Check DHIRES switch status
	$C07F	Write: Disable double hi-res switch
	$C07E	Write: Enable double hi-res switch
	$C05E	OFF: Double hi-res on
	$C05F	ON: Double Hi-res off

— Auxiliary Bank Memory—

Auxiliary memory switches determine whether memory will be read from or written to main or auxiliary 64K RAM banks or a combination of both. The three switches controlling auxiliary memory are identified in Table 4.3.

As the switches in Table 4.3 and Figure 4.4 indicate, the 64K memory map that the 65C02 sees can consist exclusively of memory from the 64K main bank, or 64K all from the auxiliary bank, or approximately 16K (zero and stack pages from $0 to $1FF and Language Card RAM from $D000-$FFFF) from one bank and approximately 48K ($200-$BFFF) from another bank.

The 65C816 in emulation mode needs to toggle the auxiliary memory management switches to address any bank of memory mapped into the auxiliary memory area of the //c.

When the 65C816 is in 16-bit native mode, it can address any and all locations in auxiliary memory without using any auxiliary memory soft switch. Memory in the auxiliary RAM is addressed by writing to or reading from the $8F bank in 16-bit mode.

For example, to read a byte at location $1000 in auxiliary memory, you would use a LDAL $8F 1000 command. An alternative would be to set the Data Bank register to $8F; then a LDA $1000 would bring the byte from auxiliary memory.

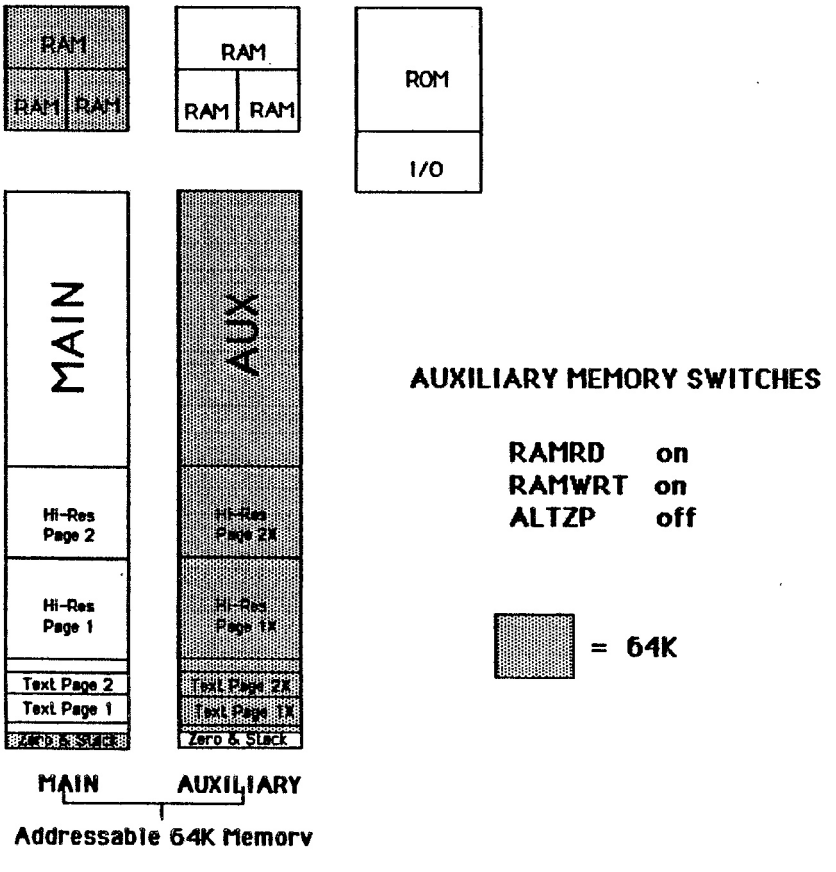
The memory management soft switches are still active and can be used to address auxiliary memory if you prefer.

Standard auxiliary memory is decoded as bank $8F of 16-bit memory because the 65C816 recognizes only one bank 0 (zero) of memory. Bank 0, the first bank of memory found by the 65C816 in native mode, is motherboard memory due to the design of the CX card. The memory bank assigned to the standard //c auxiliary bank when the 65C816 is in emulation mode is bank 0 (zero), however. Because there cannot be more than one bank 0 when the 65C816 is in 16-bit mode, the auxiliary memory is decoded as bank $8F, high enough in memory to potentially allow 8 megabytes of memory to be directly addressed.

Because the auxiliary bank of memory is used to create 80-column and double hi-res displays, this special treatment of the auxiliary bank does not adversely affect its use as 16-bit programs using this RAM would have to be guard against overrunnning the memory used to create 80-column or double hi-res displays anyway. To read or write information into this area in 16-bit mode, simply use the $8F bank designator for data access or do a long jump or subroutine call to an area in $8F xxxx.

Figure 4.4

Example of Main/Aux Combination



— Video Memory —

There are three areas of main and auxiliary memory devoted to video display: text, low-res, and hi-res display memory.

Forty-column and normal hi-res displays can come from either main or auxiliary memory depending upon the setting of video memory soft switches. Eighty-column or double hi-res displays draw memory from both main and auxiliary text display memory.

Table 4.3 shows which soft switches are involved in video display. A combination of soft switches and memory areas in both the motherboard and auxiliary areas is required to generate an 80-column display. Similar combinations are required for double hi-res graphics. The switches allow video displays whose even-column characters come from the auxiliary bank and odd-column characters come from the main bank.

When in emulation mode, the 65C816 addresses these video switches as the 6502 normally would. In native mode, these soft switches still must be addressed to determine which area of video memory is displayed. All the soft switches work as they normally do, including the 80-column and double hi-res switches.

The only difference in native mode is that the video memory located on the auxiliary area of memory may be directly written to by such commands as STAL $8F 0400. You may wish to substitute a direct address for the alternation commonly done between PAGE2 OFF ($C054) and PAGE2 ON ($C055) for writing to the alternating bytes of the interlaced 80-column or double hi-res screen.

Eighty-column and double hi-res graphic displays can still be controlled through the video memory soft switches as is normally done. You may also use COUT in the monitor to send characters to the screen if your registers remain in 8-bit size when you are in native mode.

MultiRam Memory

All 64K memory banks from the MultiRam CX memory card are brought into the standard //c memory map as one 64K bank of auxiliary memory at a time when the 65C816 is in emulation mode, shown by Figure 4.2. The memory card's different 64K memory banks are swapped into and out of the auxiliary memory area by writing the desired number of the MultiRam 64K memory bank into the bank select switch shown in Figure 4.2.

For example, if a value of 4 is written to location $C071, then bank 4 of MultiRam memory is swapped with standard bank 0 (zero) auxiliary memory and any reading or writing to locations within auxiliary memory occur in MultiRam bank 4.

MultiRam's 64K memory banks, other than the one bank treated as standard auxiliary memory discussed in the previous section, are addressed as contiguous, linear memory when the 65C816 is in 16-bit native mode as Figure 4.3 indicates. In other words, data may stretch throughout MultiRam memory without any breaks, without requiring any bank-switching to address, and the 65C816 may address any

byte in memory with simple commands such as LDAL $03 1000 (Load Accumulator long) or JSL $04 2000 (Jump to subroutine long) no matter where a program is running in memory.

— Addressing Mode Comparison —

The ability to address a large range of memory makes the 16-bit native addressing modes very compact in comparison to the emulation modes. The code involved with memory management for native mode requires fewer bytes to put a byte anywhere in a large range of memory than are required by bank-switching code when in emulation mode or when a 65C02 is left in the //c.

For example, here is a comparision between native and emulation code and associated processor cycle times required to place a single byte representing "?" in 64K bank 3 while the routine writing the byte is in motherboard memory and read/write addressing of the program is to return to motherboard memory after the one byte is written:

EMULATION MODE

	bytes	cycles
LDA #$BF a "?"	2	2
LDX #$03 bank to write to	2	2
STX $C071 select bank 3	3	4
STA $C005 RAMWRT on	3	4
STA $1000	3	4
STA $C004 RAMWRT off	3	4
	16	20

NATIVE MODE

	bytes	cycles
LDA #$BF	2	2
STAL $03 1000	4	5
	6	7

The ability to address a large range of memory also makes the 16-bit native addressing modes very fast in comparison to the emulation modes. The code involved with memory management for native mode requires fewer processor cycles to put a byte anywhere in a large range of memory, usually two or more times faster than the more involved bank-switching techniques required when in emulation mode or when a 65C02 is left in the //c. The preceeding examples show a threefold speed increase.

— Potential Problems —

You may experience problems if you try to move between emulation and native mode if the areas of memory involved during the switch are all MultiRam memory. Problems can result as the last 64K Program bank selected in 16-bit mode will

determine which bank of MultiRam auxiliary memory is on line when you return to emulation mode.

For example, if you have 8-bit code running on MultiRam auxiliary bank 2 and then enter native mode to continue with 16-bit routines in the $02 xxxx bank area but then jump to routines in the $03 xxxx area, you should not try to immediately reenter emulation mode as the auxiliary bank you would return to would be bank 3, the last 64K bank used in native mode. If no 8-bit continuation code were there, your program would crash.

You should always return in native code to the bank where the emulation continuation code lies.

An easier method than this, however, is to always do mode switching from motherboard memory. When switching from motherboard, there would be no confusion about which bank of memory will be switched in no matter which direction the switch; you are always in bank zero. Place the XCE exchange routines in motherboard and enter these routines via JML's or JSL's and end the switch routines with a JML or RTL.

Chapter 5

SOFTWARE DEVELOPMENT TOOLS

To program the 65C816, you will need to use a language that can produce 65C816 code. Such languages and other development tools for the 65C816 are now being developed. A few such development tools are currently available while others will be released in the near future.

The following list of languages shows what is available now and what will soon be available for the 65C816. The name of the company producing the language as well as contact information for that company are given. An estimated release date is also given for languages not yet available.

ASSEMBLERS

ProDOS ORCA/M 4.0

The Byte Works, Inc.
8000 Wagon Mound Dr. NW
Albuquerque, NM 87120
(505)898-8183

ORCA/M is a macro assembler for the 65C02, 65C802, and 65C816 to be used on an Apple // series computer running ProDOS. ORCA/M was the first assembler to support the 65C816 and additional support for the 65C816 is being added. Long addressing modes are fully supported by the assembler. ORCA source code is available for a seperate charge to allow you to easliy customize the program. An earlier DOS 3.3 version supporting the 65C816 was also available. ORCA/M is a sophisticated assembler that features a full screen editor, macros, and a host system that can be used by higher level languages. ORCA/M is not copy protected. ORCA/M is an excellent assembler for large scale projects due to its sophistication and flexibility.

MERLIN PRO

Roger Wagner Publishing
10761 Woodside Ave., Ste E
Santee, CA 92071
(619) 562-3670

Merlin Pro is a macro assembler for the 65C02, 65C802, and 65C816 and can be used on an Apple // series computer running DOS 3.3 or ProDOS. Support for the 65C802 and 65C816 is available only with newer versions of Merlin, at least version 2.4 or higher. Newer versions of Merlin with a full screen editor and additional 65C816 support are planned. Merlin does not currently support 65C816 long addressing but macros can be easily constructed to handle long addressing.

S-C MACRO ASSEMBLER V2.0

S-C Software Corporation
PO Box 280300
Dallas, TX 75228
(214)324-2050

The S-C Macro Assembler is a macro assembler for the 65C02, 65C802, and 65C816 to be used on an Apple // series computer running DOS 3.3 or ProDOS 1.1.1. Older versions do not support the 65C816. Long addressing is directly supported with up to 32-bit expressions. The assembler uses a line editor with an optional cost screen editor. The program is unprotected and a free issue of "S-C Assembly Lines", which does offer 65C816 tips, comes with purchase of the program.

LISA 3.2 MACRO ASSEMBLER

Lazerware
2271 Indian Horse Dr.
Norco, CA 91760
(714)735-1041

Lisa supports the 65C802 and 65C02. Contact the publisher for more information.

HIGH LEVEL LANGUAGES

BASIC

Basic 16

Arrays, Inc.
6711 Valjean Avenue
Van Nuys, CA 91406
(213) 410-9466

Basic 16 is a 65C802 and 65C816 version of Basic upwardly compatible from Applesoft Basic. Basic 16 features an enhanced emulation mode disassembler and 16-bit extended version of floating point Basic with named subroutines, if-then-else statements, enhanced graphic and windowing commands, 16-bit math routines and more. Already existing Applesoft programs can be run on Basic 16 and can be easily modified to take advantage of its new features. Basic 16 programs will also be able to take full advantage of MultiRam's larger memory due to the 65C816's ability to directly address up to 16 megabytes. Basic 16 will come with a manual as well as tutorial disks with sample Basic 16 programs. Basic 16 will use either the DOS 3.3 or ProDOS operating systems. Basic 16 will be available by early 1986.

Part One: Basics

Chapter 1: Basic Assembly Language Programming Concepts

A brief review of binary and hex representation of numbers, ASCII representation of characters, boolean logic, signed numbers, binary coded decimal, computer arithmetic, machine language, assembly language, higher level languages, basic programming concepts.

Part Two: Architecture

Chapter 2: Architecture of the 6502

The 6502, the basic 65x machine -- microprocessor architecture -- the 6502 registers (A, X, Y, P, SP, PC) -- Status flags -- addressing modes -- instructions -- Pipelining -- Memory order of multiple-byte values -- Memory-mapped I/O -- Interrupts -- NMOS process -- Bugs and quirks

Chapter 3: Architecture of the 65C02

65C02 architecture -- Additional addressing modes -- Instructions -- CMOS process -- Bugs and quirks

Chapter 4: Sixteen-Bit Architecture: The 658816 and the 65802

Enhancements on all fronts -- Power-on status: 6502 Emulation Mode -- 65816 Native Mode -- Registers (K, B, D, A, X, Y, P) -- Switching registers between 8 and 16 bits -- 6502/65C02 addressing modes on the 65816 -- New addressing modes -- Instructions -- Interrupts -- 65802 Native Mode -- Emulation Mode -- Switching between Emulation & Native Modes -- Bugs and quirks

Part Three: Tutorial

Chapter 5: SEP, REP and Other Details

SEP and REP -- Assembler used in this book -- Address notation

Chapter 6: First Examples: Moving Data

Load and Store -- Push and pull -- Transfer and Exchange -- Storing zero to memory -- Block moves

Chapter 7: The Simple Addressing Modes

Addressing Modes -- Immediate -- Absolute -- Direct Page -- Indexing -- Absolute,X -- Absolute,Y -- Direct,X -- Direct,Y -- Accumulator -- Implied -- Stack -- Direct Page Indirect -- Absolute long -- Absolute long,X -- Direct page indirect long -- Block move

Kyan Pascal

Kyan Software
1850 Union Street, Suite 183
San Francisco, CA 94123
(415) 775-2923

Kyan Pascal is curently being shipped for the 65C02. A full featured 65C816 version supporting memory available on MultiRam memory cards will be available in early to mid-1986. Kyan Pascal is unprotected and uses the ProDOS operating system. Kyan Pascal 16-bit programs will be able to directly use the memory available on MultiRam memory cards. Kyan 16-bit Pascal will compile to native 65C816 code, not p-code, for very rapid run-time program execution. Compilation will also be extremely fast. Conversion of Apple Pascal 1.1 and 1.2 source code to Kyan 16-bit Pascal will be possible. Kyan 16-bit Pascal will be easy to use, will be a full Pascal implementation, will have a built-in full screen editor, assembler, and will include powerful extensions.

PROGRAMMING REFERENCE SOURCES

Because the 65C816 is a new processor, there are currently no reference books available on programming the 65C816.

The first programming reference book on the 65C816 will be available by Decemeber of 1985, Programming the 65C816 (Including the 6502, 65C02, and 65802). The book's publisher is Brady Computer Books, a division of Simon and Schuster. David Eyes and Ron Lichty are the book's co-authors. A seperate disk containing programming examples and the program DEBUG16, a 65C816 mini-assembler and disassembler, will be available seperately from the publisher.

Following is the book's table of contents reprinted with permission of Brady Computer Books, a division of Simon and Schuster.

Programming The 65C816

Table of Contents

Part One: Basics

Chapter 1: Basic Assembly Language Programming Concepts

A brief review of binary and hex representation of numbers, ASCII representation of characters, boolean logic, signed numbers, binary coded decimal, computer arithmetic, machine language, assembly language, higher level languages, basic programming concepts.

Part Two: Architecture

Chapter 2: Architecture of the 6502

The 6502, the basic 65x machine -- microprocessor architecture -- the 6502 registers (A, X, Y, P, SP, PC) -- Status flags -- addressing modes -- instructions -- Pipelining -- Memory order of multiple-byte values -- Memory-mapped I/O -- Interrupts -- NMOS process -- Bugs and quirks

Chapter 3: Architecture of the 65C02

65C02 architecture -- Additional addressing modes -- Instructions -- CMOS process -- Bugs and quirks

Chapter 4: Sixteen-Bit Architecture: The 658816 and the 65802

Enhancements on all fronts -- Power-on status: 6502 Emulation Mode -- 65816 Native Mode -- Registers (K, B, D, A, X, Y, P) -- Switching registers between 8 and 16 bits -- 6502/65C02 addressing modes on the 65816 -- New addressing modes -- Instructions -- Interrupts -- 65802 Native Mode -- Emulation Mode -- Switching between Emulation & Native Modes -- Bugs and quirks

Part Three: Tutorial

Chapter 5: SEP, REP and Other Details

SEP and REP -- Assembler used in this book -- Address notation

Chapter 6: First Examples: Moving Data

Load and Store -- Push and pull -- Transfer and Exchange -- Storing zero to memory -- Block moves

Chapter 7: The Simple Addressing Modes

Addressing Modes -- Immediate -- Absolute -- Direct Page -- Indexing -- Absolute,X -- Absolute,Y -- Direct,X -- Direct,Y -- Accumulator -- Implied -- Stack -- Direct Page Indirect -- Absolute long -- Absolute long,X -- Direct page indirect long -- Block move

Chapter 8: The Flow of Control

Jump instructions -- Long branches and jumps -- Conditional branching -- Limitations of conditional branches -- Unconditional branches

Chapter 9: Built-in Arithmetic Functions

Increment and decrement -- Addition and subtraction: unsigned arithmetic -- Comparison -- Signed arithmetic -- Signed comparisons - Decimal mode

Chapter 10: Logic and Bit Manipulation Operations

Logic functions: AND, OR, EOR, complement - Bit Manipulation -- Logical shifts and rotates

Chapter 11: The Complex Addressing Modes

Relocating the direct page -- Direct page indirect indexed -- Direct page indexed indirect -- Absolute indexed indirect -- DP indirect long indexed -- Stack relative -- Stack relative indirect indexed -- Push effective

Chapter 12: The Basic Building Block: The Subroutine

Subroutine calls -- Returning from subroutines -- Long JSR's -- Branch to subroutine -- When to use a subroutine: Negation -- Parameter passing

Chapter 13: Interrupts and System Control Instructions

Interrupts, RTI, and BRK -- Polling of interrupts -- Interrupt response time -- Status register control instructions

Part Four: Applications

Chapter 14: Selected Code Samples

Examples of different types of 65816 code -- Compiler-generated code -- Example of an assembly-coded recursive subroutine -- Multiply and Divide: Converting from 6502 to 65C02 to 65802 -- Calling a 6502 routine from a native mode program -- Processor identification routine -- The sieve benchmark

Chapter 15: DEBUG16: A 65816 Programming Tool

A guided tour through DEBUG16, a 65816 step-and-trace debugger -- DEBUG16 code, routine by routine, and how it works

Chapter 16: Design and Debugging

Common bugs and solutions -- Is d flag set incorrectly? -- Limit subroutines to be called only from a certain mode? -- Dealing with modes systematically -- Adjusting the carry flag before adding and subtracting -- And so on

Design -- Top-down design and structured programming -- Documentation

Part Five: Reference

Chapter 17: The Addressing Modes

Each of the 6502, 65C02, 65802 and 65816 addressing modes, each beginning a new page, each graphically drawn to illustrate its use

Chapter 18: The Instruction Sets

The 6502, 65C02, 65802, and 65816 opcodes, each beginning a new page, listed alphabetically, each with description, including differences between eight- and sixteen-bit modes; flags affected; addressing modes available; opcodes; syntax examples; and byte and cycle counts

Chapter 19: Instruction Lists

Alphabetical list -- Functional list -- Numerical list -- Opcode map

Appendices

Appendix A: 65x Signal Description

Signal description of 6502, 65C02, 65802, and 65816 pinouts and how they relate to system implementation

Appendix B: 65x Series Support Chips

A brief introduction to some of the available companion parts, with examples: -- the 6551: serial I/O -- the 6521: parallel I/O

Appendix C: The Rockwell 65C02

A part with the same name, but extra instructions -- bit manipulation for control applications -- incompatability with the 65802 and 65816 family expansion -- the extra instructions, with descriptions, how they affect flags, addressing modes, and byte and cycle counts

Appendix D: 65x Instruction Groups

Dividing the 65x instructions into groups based on their actions, their addressing modes, and their bit patterns.

Appendix E: 65816 Data Sheet

Pinouts -- Timing diagrams -- Compatability issues -- Signal descriptions -- Assembler syntax description

Appendix F: The ASCII Character Set

Complete ASCII chart, with high bit both set and reset

Other books on programming the 65C816 will become available in 1986. Another detailed reference book on the 65C816 will be available in Spring of 1986.

Articles on programming the 65C802 and the 65C816 are starting to appear in Apple publications that offer assembly language programming assistance, such as "Apple Assembly Line" by S-C Software, and more will appear in the near future.

A 65C816 information data sheet giving more engineering details on the 65C816 is available from: Western Design Center, 2166 East Brown Road, Mesa, AZ 85203.

Chapter 6

M A X - O S and V I P P r o f e s s i o n a l

To give you the optimum use of your new MultiRam 65C816 option, Checkmate Technology offers two new 65C816 programs: MAX-OS and VIP Professional.

M A X - O S

MAX-OS is a high-performance Unix-like operating system written by Micro Magic Inc. of Millersvillle, Maryland for the 65C816 microprocessor.

The operating system, compatible with all the most popular Apple // peripherals, will execute in full 16-bit mode on MultiRam 65C816 equipped Apple //e's and //c's.

While application programs can be written in 16-bit code and still use any existing Apple // operating system due to the 65C816's compatibility with the 65C02 processor, existing operating systems do not take advantage of the 65C816's advanced capabilities. MAX-OS will be the first operating system to make full use of the 65C816's advanced operation.

Written exclusively in 65C816 code, MAX-OS, while having powerful Unix-like structure and operation, is easy to use for even the novice. Complete documentation thoroughly explains how to quickly use MAX-OS.

Programmers will appreciate its power and elegance as MAX-OS is very compact ; MAX-OS uses only approximately 16K of memory but can address and utilize all the memory available on MultiRam memory cards. Programmers will also appreciate its easy customization allowing drivers to be added for different applications.

MAX-OS includes a File Management Subsystem, Command String Interpreter Subsystem, Input/Output Driver Subsystem, and various disk based utilities such as file transfer and formatting routines. Also included are provisions for interrupt driven device handlers, memory expansion, and the addition of general utilities. MAX-OS provides an effective resource management structure to allow multitasking and task scheduling assignment.

MAX-OS features include:

-Friendly user environment
-Unix-like structure
-Multiprogramming operation
-Disk based utilities, saving memory space
-Support for Apple compatible peripherals
-Access to operating system file and peripheral routines
-Support for interrupt driven devices
-Support for multitasking

MAX-OS will come with full documentation detailing its commands, a memory map showing its utilization of MultiRam enhanced //c and //e memory, entry points

for user customization, source code for input/output drivers, and complete instructions for easy customization.

MAX-OS uses ProDOS compatible disk formats and files structures to allow existing ProDOS files to be transfered to MAX-OS. Like ProDOS, MAX-OS uses a hierarchial file structure, can support file sizes up to 16 megabytes, has fast disk access times, and is device independent so that large storage devices such as hard disk drives are easily supported.

MAX-OS provides a full development environment which is responsive but not restrictive. High level languages now being developed for the 65C816 can best use MAX-OS for optimum performance.

MAX-OS will be shipped with each MultiRam 65C816 option by January 1986. You will automatically receive your copy of MAX-OS when it is first released by simply returning your Warranty Registration card.

VIP PROFESSIONAL FOR THE MULTIRAM 65C816

VIP Professional, by VIP Technologies, is an integrated productivity powerhouse designed for the 16-bit MultiRam 65C816: spreadsheet analysis, information management, and colorful business graphics — all three in one integrated program.

VIP Professional is modeled after the powerful, best-selling Lotus 1-2-3 program. It has every feature, every command of Lotus 1-2-3 on the IBM PC. The experienced Lotus 1-2-3 user will feel right at home using the same keystrokes and macros used with 1-2-3. In fact, Lotus 1-2-3 files can be transfered to VIP Professional on your Apple, worked with, and then transferred back to the IBM. Files can also be transferred between VIP Professional running on an IBM, Amiga, Macintosh, Atari ST, or any IBM compatible computer.

And thanks to MultiRam's 65C816, VIP Professional is as fast if not faster than Lotus 1-2-3 on the IBM PC.

VIP Professional is as easy to use as it is powerful with Macintosh delights like those in Jazz — a mouse, icons and drop down menus -- so that a beginner can use it right away. No more learning/GF or (ESC) GR99:C248 and other VisiCalc and Multiplan cryptic mysteries. Just point your mouse and click to make any change or do anything! VIP Professional will do the learning for you.

If you have questions about any feature or command, you can get instant help. Just point at the menu and click the mouse twice. You can find out how to do whatever you are doing, or more about other parts of the program. A thorough reference manual is also available as well as a tutorial for the novice.

VIP Professional uses the advanced double hi-res capabilities of the Apple. This means optional character fonts and graphs in 16 colors. VIP Professional also takes full advantage of MultiRam memory cards accessing up to 512K on the MultiRam //c card and up to 4 megabytes on the MultiRam //e memory cards.

Some specific information on the three parts of VIP Professional:

1) Spreadsheet

Financial planning, sophisticated business inventory control, budget or modeling, home budgeting -- VIP Professional will fill every need. The power user has a full 8192 row by 256 column worksheet to use with up to four megabytes of memory on the fully expanded MultiRam //e 65C816 version.

VIP Professional will even read in VisiCalc or Multiplan files and then teach them a few new tricks, like putting them in a graph, or using them as part of a database.

2) Database

VIP Professional includes a powerful data manager to arrange, store and analyze your important information. It can be used to store records about your home or business, and do extensive searches, sorts and comparisons. It can use the data in your spreadsheet. It allows up to 8192 records, with up to 256 fields. Queries can be made with up to 32 search criteria. With its powerful statistical functions, it is a real workhorse.

3) Presentation Graphics

VIP Professional gives shape to your figures by letting you graph your data into six different types of graphs, in 16 hi-res colors -- pie charts, bar graphs, stacked bar graphs, line graphs, scatter graphs and X/Y graphs. When you want to show a graph of your worksheet or database data, for instant professional presentations just click the mouse and it's there. You can re-create new graphs instantly when you change data. When you're ready, you can print your graphs in a variety of styles and fonts, even in color.

VIP Professional not only meets but out-features Lotus 1-2-3:

Feature	VIP	1-2-3
123 Commands	Yes	Yes
123 Graphics	Yes	Yes
123 Database	Yes	Yes
123 Macros	Yes	Yes
123 Files	Yes	Yes
Enhanced Graphs	Yes	No
Uses mouse	Yes	No
Drop-Down Menus	Yes	No
Icons	Yes	No
Multiple windows	Yes	No
Available on Apple	Yes	No
Easy to use	Yes	No
Affordable	Yes	No

VIP Professional for the MultiRam 65C816 option decidedly gives the professional touch to your Apple.

MultiRam 65C816 owners can get the 16-bit version of VIP Professional, which is over four times as fast as the 8-bit version, at a 35 percent lower price than the 8-bit version with the special coupon enclosed in this manual. See the coupon for details. (If there is no coupon in your manual, just drop us a note with your Warrany Registration card and we will send you a coupon.)

VIP Professional for the MultiRam 65C816 option will be available by February of 1986. You will be notified when VIP Professional is ready. See your dealer for your MultiRam discount on VIP Professional.

Chapter 7

THE WARRANTY

The MultiRam 65C816 CX Kit, like all of Checkmate Technology's peripheral cards, carries a 5 YEAR LIMITED WARRANTY.

CHECKMATE TECHNOLOGY, INC. warrants products against defects in material and workmanship for a period of 5 years (90 days for software) after purchase. During the warranty period, CHECKMATE TECHNOLOGY, INC. will at its option, repair or replace at no charge, or refund the purchase price to the purchaser, products that prove to be defective provided that the CHECKMATE TECHNOLOGY product is returned, shipping prepaid, to CHECKMATE TECHNOLOGY, INC. For products returned by other than U.P.S., the sender assumes greater risk of loss and delays.

Your sales receipt is your warranty validation. The receipt must be provided when requesting warranty work to be performed unless a valid product registration card is on file at the company headquarters. This warranty does not apply if, in the sole opinion of CHECKMATE TECHNOLOGY, INC., the product has been damaged due to abuse, misuse, misapplication, accident or as a result of service or modification by other than an authorized CHECKMATE TECHNOLOGY, INC. service center.

THIS WARRANTY IS EXPRESSLY IN LIEU OF ANY AND ALL WARRANTIES EXPRESSED OR IMPLIED. CHECKMATE TECHNOLOGY DOES NOT WARRANT THAT GOODS ARE MERCHANTABLE OR FIT FOR ANY PARTICULAR PURPOSE. CHECKMATE TECHNOLOGY SHALL NOT BE HELD RESPONSIBLE FOR ANY CONSEQUENTIAL DAMAGES OR LOSSES ARISING FROM THE USE OF THIS PRODUCT.

This warranty applies to CHECKMATE TECHNOLOGY products purchased in the United States. The warranty may vary for products purchased outside the continental United States. Contact CHECKMATE TECHNOLOGY, INC. for warranty service information.

SECOND HAND OWNERSHIP

Purchasers of used Checkmate Technology products should register the transfer with Checkmate Technology. The cost of registration is $10.00 and is a wise investment: any remaining unexpired warranty period will be transferred into your name and you will receive newsletters providing product information, including news of product updates.

To transfer the unexpired warranty period, the original owner must have filled out and returned the Warranty Registration card to Checkmate Technology at the time of purchase.

If the original owner failed to register the card, for a one-time fee of $30.00 you may transfer the unexpired warranty period (based on either the original purchase receipt showing date of purchase or by date of manufacture indicated by serial number on the card). Checkmate Technology reserves the right to decline any warranty transfer.

In order to transfer registration you must provide Checkmate Technology with:

1. Product name
2. Serial number on the card
3. Name of previous owner (sellers name)
4. Your name, address, and telephone number
5. Date of sale
6. Your check for $10 ($30 if the card was not registered)

EXTENDING THE WARRANTY

If you would like to extend the warranty on your newly purchased or used card, Checkmate Technology offers an EXTENDED WARRANTY PROGRAM.

New owners can take advantage of an additional five year warranty by sending a check for $25.00 together with the product Warranty Registration card. This offer is limited to new owners and must be sent to the company within thirty days of purchase.

Second hand owners can extend the unexpired warranty to a maximum of five years. The cost of extending the warranty is $15.00 per year in addition to the transfer registration fee. For example, if there is only one year left on the unexpired warranty, a check for $70.00 would be sent: $60.00 for 4 years plus a $10 registration fee ($30 if the original owner did not register the card). Checkmate Technology reserves the right to limit the duration it accepts for extended warranties.

Chapter 8

SERVICE POLICIES

This chapter explains Checkmate Technology's repair and product update policies and other pertinent customer information.

HOW TO GET HELP

If you are unable to get your MultiRam CX card with the 65C816 option to work properly after testing, please contact the Customer Service Department. We will try to solve your problem as quickly as possible.

Telephone Help

Before sending in your MultiRam CX card with the 65C816 option, you may want to call a Customer Service representative. Some problems may be solved without returning your card.

The telephone number and hours for technical assistance are:

Customer Service: (602) 966-5802
Hours: Mon. to Fri., 9am to 5pm (Mountain Standard Time)

If you need to call us, please have the following information available to save time:

1. Nature of the problem with the card.
2. Serial number of the Checkmate Technology card.
3. Serial number of the Apple //c used (found on the bottom of the //c case).
4. Peripherals and program in use when the problem occurred.

If your card needs to be returned to the factory for repair, you will be issued a Return Merchandise Authorization number during your call. Write this number on the outside of the package that you will use to ship your card.

Telephone Policy

We want to be as helpful as possible with your questions and problems. We understand that the problem you are having is important, and we will provide technical assistance to try to answer it. We ask that you use our Customer Service telephone number for assistance. Support is not available through the toll-free order desk telephone number; collect calls are not accepted by Customer Service.

HOW TO RETURN A CARD

If it is necessary to return your MultiRam CX card, follow these steps:

1. Please wrap your card in anti-static material. If none is available, wrap the card in aluminum foil. Returning a card without wrapping it in protective anti-static material or aluminum foil may cause further damage to the card and will VOID YOUR WARRANTY. Insert the pins on the bottom of the card into the anti-static foam shipped with the card to protect the pins from damage.

2. Include a note with your name, address, home telephone number, a daytime telephone number, plus:

   A. Detailed description of the problem.
   B. Serial number of the Apple //c used (found on the bottom of the case).
   C. Peripherals and program in use when the problem occurred.

3. If you are a registered owner, meaning that you completed and returned your warranty registration card to Checkmate Technology, you are not required to send in proof of purchase. If you have not registered your MultiRam card and 65C816 option, include proof of purchase.

NOTE: ANY CARD RETURNED WITHOUT PROOF OF PURCHASE, OR WHICH IS NOT REGISTERED, WILL BE TREATED AS A PRODUCT OUT OF WARRANTY AND WILL BE CHARGED FOR NON-WARRANTY REPAIR AT RATES THEN IN EFFECT.

4. Pack the wrapped card, note, and proof of purchase (if needed) in a sturdy box and cushion the card with non-static packing material such as crumpled newspapers.

5. Ship the card, postage pre-paid, to:

   Checkmate Technology, Inc.
   511 South Rockford Drive
   Tempe, Az. 85281-3021
   Attn: Repair Dept.

Checkmate Technology, Inc. assumes no responsibility for cards either damaged or lost in shipping to Checkmate Technology for service. Please insure your valuable card. The cost of shipping insurance is low compared to replacing a lost card. We suggest U.P.S. as they insure the first $100.00 at no charge and require the addressee to sign for each package.

RETURNING YOUR REPAIRED CARD

After repair or replacement, your MultiRam 65C816 card will be shipped to you in the U.S.A. by U.P.S., regular delivery, postage paid. Returns outside the U.S.A. may require a handling and shipping charge. Contact us regarding repairs outside the U.S.A.

REPLACING DAMAGED DISKETTES

All Checkmate Technology disks are warrantied for 90 days and will be replaced free of charge if a defect is discovered within the warranty period.

If you damage or wear-out a CHECKMATE TECHNOLOGY, INC. program disk after the 90 day warranty period and are a REGISTERED OWNER, we will replace your old program diskette for $15.00, return postage prepaid. Non-registered software owners will be required to pay $25.00. The additional ten dollars covers the costs of setting up a new owner registration.

Return the damaged or worn-out program diskette to the Customer Service Department, along with a note detailing the problem, and a replacement disk will be sent to you.

SOFTWARE UPDATES

Checkmate Technology may from time to time upgrade the standard features of the MultiRam software. Registered owners will be notified of updates through our newsletter or special mailings from their dealers.

To receive an updated software version, you may return to your dealer with blank disks and receive a free update or send $6.00 to Checkmate Technology to cover disk, shipping and handling costs, and updated disks will be sent to you. You will not need to return the older software version.

NEWSLETTERS

Checkmate Technology will publish informative newsletters for registered owners of its products. "Check-It-Out" Newsletters will contain information on updates to products, information on modifications to software, general information on product usage, and news of product developments at Checkmate Technology, Inc.

If you would like to share any discoveries, programming tips, comments, or suggestions with other owners, please write to us and we will include them in the newsletter.

INDEPENDENT SOFTWARE DEVELOPERS

We actively solicit your help in converting existing Apple programs to use the powerful MultiRam CX 512K and 65C816 combination and in publishing new 65C816 programs. Anyone writing an original software package featuring the MultiRam card or modifying an existing program is invited to call to arrange a product review. Please do not send your original software without first contacting us. Call the Customer Service Department for further details on software submissions.

SUGGESTIONS

If you have suggestions, comments, or questions regarding Checkmate's products , please contact us. Address correspondence to:

CHECKMATE TECHNOLOGY, INC.
509 South Rockford Drive
Tempe, Arizona 85281-3021

Attn: Andrew P. Niemic President

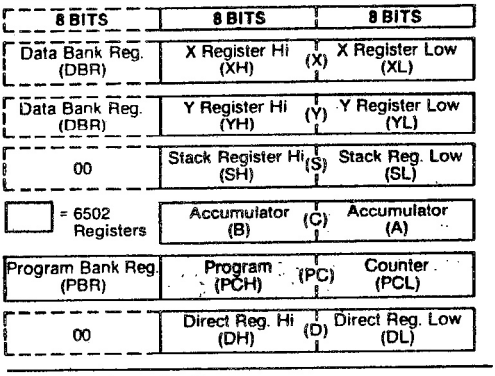
Our goal is a satisfied customer!

APPENDICES

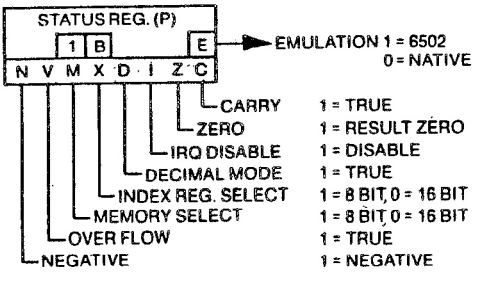
The 65C816 specifications and tables presented in the following Appendices are derived from Western Design Center Data sheets and are reprinted with permission of Western Design Center.

APPENDIX A

W65C816 Processor Programming Model



Status Register Coding



APPENDIX B

Internal Registers

Accumulators (A, B, C)
The Accumulator is a general purpose register which stores one of the operands, or the result of most arithmetic and logical operations. In the Native mode (E=0), when the Accumulator Select Bit (M) equals zero, the Accumulator is established as 16 bits wide (A + B = C). When the Accumulator Select Bit (M) equals one, the Accumulator is 8 bits wide (A). In this case, the upper 8 bits (B) may be used for temporary storage in conjunction with the Exchange Accumulator (XBA) instruction.

Data Bank Register (DBR)
During modes of operation, the 8-bit Data Bank Register holds the default bank address for memory transfers. The 24-bit address is composed of the 16-bit instruction effective address and the 8-bit Data Bank address. The register value is multiplexed with the data value and is present on the Data/Address lines during the first half of a data transfer memory cycle for the W65C816. The Data Bank Register is initialized to zero during Reset.

Direct (D)
The 16-bit Direct Register provides an address offset for all instructions using direct addressing. The effective bank zero address is formed by adding the 8-bit instruction operand address to the Direct Register. The Direct Register is initialized to zero during Reset.

Index (X and Y)
There are two Index Registers (X and Y) which may be used as general purpose registers or to provide an index value for calculation of the effective address. When executing an instruction with indexed addressing, the microprocessor fetches the opcode and the base address, and then modifies the address by adding the Index Register contents to the address prior to performing the desired operation. Pre-indexing or post-indexing of indirect addresses may be selected. In the Native mode (E=0), both Index Registers are 16 bits wide (providing the Index Select Bit (X) equals zero). If the Index Select Bit (X) equals one, both registers will be 8 bits wide, and the high byte is forced to zero.

Processor Status (P)
The 8-bit Processor Status Register contains status flags and mode select bits. The Carry (C), Negative (N), Overflow (V), and Zero (Z) status flags serve to report the status of most ALU operations. These status flags are tested by use of Conditional Branch instructions. The Decimal (D), IRQ Disable (I), Memory/Accumulator (M), and Index (X) bits are used as mode select flags. These flags are set by the program to change microprocessor operations.

The Emulation (E) select and the Break (B) flags are accessible only through the Processor Status Register. The Emulation mode select flag is selected by the Exchange Carry and Emulation Bits (XCE) instruction. Table 1, W65C802 and W65C816 Mode Comparison, illustrates the features of the Native (E=0) and Emulation (E=1) modes. The M and X flags are always equal to one in the Emulation mode. When an interrupt occurs during the Emulation mode, the Break flag is written to stack memory as bit 4 of the Processor Status Register.

Program Bank Register (PBR)
The 8-bit Program Bank Register holds the bank address for all instruction fetches. The 24-bit address consists of the 16-bit instruction effective address and the 8-bit Program Bank address. The register value is multiplexed with the data value and presented on the Data/Address lines during the first half of a program memory read cycle. The Program Bank Register is initialized to zero during Reset. The PHK instruction pushes the PBR register onto the Stack.

Program Counter (PC)
The 16-bit Program Counter Register provides the addresses which are used to step the microprocessor through sequential program instructions. The register is incremented each time an instruction or operand is fetched from program memory.

Stack Pointer (S)
The Stack Pointer is a 16-bit register which is used to indicate the next available location in the stack memory area. It serves as the effective address in stack addressing modes as well as subroutine and interrupt processing. The Stack Pointer allows simple implementation of nested subroutines and multiple-level interrupts. During the Emulation mode, the Stack Pointer high-order byte (SH) is always equal to one. The bank address for all stack operations is Bank zero.

APPENDIX C

W65C802 and W65C816 Instruction Set—Alphabetical Sequence

ADC	Add Memory to Accumulator with Carry
AND	"AND" Memory with Accumulator
ASL	Shift One Bit Left, Memory or Accumulator
BCC	Branch on Carry Clear ($P_C = 0$)
BCS	Branch on Carry Set ($P_C = 1$)
BEQ	Branch if Equal ($P_Z = 1$)
BIT	Bit Test
BMI	Branch if Result Minus ($P_N = 1$)
BNE	Branch if Not Equal ($P_Z = 0$)
BPL	Branch if Result Plus ($P_N = 0$)
BRA	Branch Always
BRK	Force Break
BRL	Branch Always Long
BVC	Branch on Overflow Clear ($P_V = 0$)
BVS	Branch on Overflow Set ($P_V = 1$)
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear Interrupt Disable Bit
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
COP	Coprocessor
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory or Accumulator by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EOR	"Exclusive OR" Memory with Accumulator
INC	Increment Memory or Accumulator by One
INX	Increment Index X by One
INY	Increment Index Y by One
JML	Jump Long
JMP	Jump to New Location
JSL	Jump Subroutine Long
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)
MVN	Block Move Negative
MVP	Block Move Positive
NOP	No Operation
ORA	"OR" Memory with Accumulator
PEA	Push Effective Absolute Address on Stack (or Push Immediate Data on Stack)
PEI	Push Effective Indirect Address on Stack (add one cycle if $DL \neq 0$)
PER	Push Effective Program Counter Relative Address on Stack
PHA	Push Accumulator on Stack
PHB	Push Data Bank Register on Stack
PHD	Push Direct Register on Stack
PHK	Push Program Bank Register 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
PLB	Pull Data Bank Register from Stack
PLD	Pull Direct Register from Stack
PLP	Pull Processor Status from Stack
PLX	Pull Index X from Stack
PLY	Pull Index Y form Stack
REP	Reset Status Bits
ROL	Rotate One Bit Left (Memory or Accumulator)
ROR	Rotate One Bit Right (Memory or Accumulator)
RTI	Return from Interrupt
RTL	Return from Subroutine Long
RTS	Return from Subroutine
SBC	Subtract Memory from Accumulator with Borrow
SEC	Set Carry Flag
SED	Set Decimal Mode
SEI	Set Interrupt Disable Status
SEP	Set Processor Status Bite
STA	Store Accumulator in Memory
STP	Stop the Clock
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
TCD	Transfer C Accumulator to Direct Register
TCS	Transfer C Accumulator to Stack Pointer Register
TDC	Transfer Direct Register to C Accumulator
TRB	Test and Reset Bit
TSB	Test and Set Bit
TSC	Transfer Stack Pointer Register to C Accumulator
TSX	Transfer Stack Pointer Register to Index X
TXA	Transfer Index X to Accumulator
TXS	Transfer Index X to Stack Pointer Register
TXY	Transfer Index X to Index Y
TYA	Transfer Index Y to Accumulator
TYX	Transfer Index Y to Index X
WAI	Wait for Interrupt
WDM	Reserved for Future Use
XBA	Exchange B and A Accumulator
XCE	Exchange Carry and Emulation Bits

APPENDIX D

Instruction Set

W65SC816 Instructions (256 OP Codes)

A. The Original 6502 Instruction Set (151 Op Codes)

1	ADC	Add Memory to Accumulator with Carry
2	AND	"AND" Memory with Accumulator
3	ASL	Shift Left One Bit (Memory or Accumulator)
4	BCC	Branch on Carry Clear
5	BCS	Branch on Carry Set
6	BEQ	Branch on Result Zero
7	BIT	Test Bits in Memory with Accumulator
8	BMI	Branch on Result Minus
9	BNE	Branch on Result Not Zero
10	BPL	Branch on Result Plus
11	BRK	Force Break
12	BVC	Branch on Overflow Clear
13	BVS	Branch on Overflow Set
14	CLC	Clear Carry Flag
15	CLD	Clear Decimal Mode
16	CLI	Clear Interrupt Disable Bit
17	CLV	Clear Overflow Flag
18	CMP	Compare Memory and Accumulator
19	CPX	Compare Memory and Index X
20	CPY	Compare Memory and Index Y
21	DEC	Decrement Memory by One
22	DEX	Decrement Index X by One
23	DEY	Decrement Index Y by One
24	EOR	"Exclusive-or" Memory with Accumulator
25	INC	Increment Memory by One
26	INX	Increment Index X by One
27	INY	Increment Index Y by One
28	JMP	Jump to New Location
29	JSR	Jump to New Location Saving Return Address
30	LDA	Load Accumulator with Memory
31	LDX	Load Index X with Memory
32	LDY	Load Index Y with Memory
33	LSR	Shift One Bit Right (Memory or Accumulator)
34	NOP	No Operation
35	ORA	"OR" Memory with Accumulator
36	PHA	Push Accumulator on Stack
37	PHP	Push Processor Status on Stack
38	PLA	Pull Accumulator from Stack
39	PLP	Pull Processor Status from Stack
40	ROL	Rotate One Bit Left (Memory or Accumulator)
41	ROR	Rotate One Bit Right (Memory of Accumulator)
42	RTI	Return from Interrupt
43	RTS	Return from Subroutine
44	SBC	Subtract Memory from Accumulator with Borrow
45	SEC	Set Carry Flag
46	SED	Set Decimal Mode
47	SEI	Set Interrupt Disable Status
48	STA	Store Accumulator in Memory
49	STX	Store Index X in Memory
50	STY	Store Index Y in Memory
51	TAX	Transfer Accumulator to Index X
52	TAY	Transfer Accumulator to Index Y
53	TSX	Transfer Stack Pointer to Index X
54	TXA	Transfer Index X to Accumulator
55	TXS	Transfer Index X to Stack Register
56	TYA	Transfer Index Y to Accumulator

B. New W65SCXXX Instructions (13 Op Codes)

1	BRA	Branch Relative always
2	PLX	Pull X from Stack
3	PLY	Pull Y from Stack
4	PHX	Push X on Stack
5	PHY	Push Y on Stack
6	STZ	Store Zero in Memory (Direct, Direct, X, Abs, Abs, X)
7	TRB	Test and Reset Memory Bits Determined by Accumulator A (Direct and Absolute)
8	TSB	Test and Set Memory Bits Determined by Accumulator A (Direct and Absolute)

C. New W65SCXXX Addressing Modes (14 Op Codes)

2	BIT	Test Bits in Memory with Accumulator (Direct, X, Absolute, X; Immediate).
2	DEC	Decrement (Accumulator)
3	Group I	Instructions (Direct Indirect (8 Op Codes))
4	INC	Increment (Accumulator)
5	JMP	Jump to New Location (Absolute Indexed Indirect)

APPENDIX D (continued)

D. Group I Instructions with New Addressing Modes (48 Op Codes)

- Direct Indirect Long Indexed with Y (8 Op Codes)
- Direct Indirect Long (8 Op Codes)
- Absolute Long and Absolute Long Indexed with X (16 Op Codes)
- Stack Relative (8 Op Codes)
- Stack Relative Indirect Indexed Y. (8 Op Codes)

1. ADC — Add Memory to Accumulator with Carry
2. AND — "AND" Memory with Accumulator
3. CMP — Compare Memory and Accumulator
4. EOR — "Exclusive-or" Memory with Accumulator
5. LDA — Load Accumulator with Memory
6. ORA — "Or" Memory with Accumulator
7. SBC — Subtract Memory from Accumulator with Borrow
8. STA — Store Accumulator in Memory

E. New Push and Pull Instructions (7 Op Codes)

1. PEA — Push Effective Absolute Address or Immediate Data Word on Stack
2. PEI — Push Effective Indirect Address or Direct Data Word on Stack
3. PER — Push Effective Program Counter Relative Indirect Address or Program Counter Relative Data Word on Stack
4. PLB — Pull Data Bank Register from Stack
5. PLD — Pull Direct Register from Stack
6. PHB — Push Data Bank Register on Stack
7. PHD — Push Direct Register on Stack
8. PHK — Push Program Bank Register on stack

F. Status Register Instructions (2 Op Codes)

1. REP — Reset Status Bits Defined by Immediate Byte 1 = Reset, 0 = Do not change
2. SEP — Set Status Bits Defined by Immediate Byte 1 = Set, 0 = Do not change

G. New Register Transfer Instructions (8 Op Codes)

1. TCD — Transfer C Accumulator to Direct Register D
2. TDC — Transfer Direct Register D to C Accumulator
3. TCS — Transfer C Accumulator to Stack Register
4. TSC — Transfer Stack Register to Accumulator C
5. TXY — Transfer X to Y
6. TYX — Transfer Y to X
7. XBA — Exchange B and A
8. XCE — Exchange Carry Bit C with Emulation Bit E

H. New Branch, Jump and Return Instructions (6 Op Codes)

1. BRL — Branch Relative Long Always (16 Bit Relative—32768 to + 32767) (Addressing Mode)
2. JML — Jump Indirect Long
3. JMP — Jump Absolute Long
4. JSL — Jump to Subroutine Long (Uses RTL for Return)
5. JSR — Jump to Subroutine (Indexed Indirect)
6. RTL — Return from Subroutine Long

I. New Block Move Instructions (2 Op Codes)

1. MVN — Move Block from Source (X Addressed) to Destination (Y Addressed). Block Length Defined by C. X, Y are Incremented.
2. MVP — Move Block from Source (X Addressed) to Destination (Y Addressed). Block Length Defined by C. X, Y are Decremented.

J. New Co-Processor Operations (1 Op Code)

1. COP — Co-Processor Instruction with Associated COP Vector and ABORT Input Supports Co-Processing Function i.e., Floating Point Processors, etc.

K. New System Control Instructions (3 Op Codes)

1. STP — Stop-the-clock Instruction Stops the Oscillator Input (or 02 Input) During 02 = 1. This Mode Is Released When RES Goes to a Zero. System Initialization May Be Desired; However, if After RESET One Performed an RTI, Program Execution Begins With the Instruction Following the STP Op Code in Program Sequence
2. WAI — Wait for Interrupt Pulls RDY Low and Is Cleared by IRQ or NMI Active Input
3. WDM — There is One Reserved Op Code Defined as WDM Which Will Be Used For Future Systems. The W65SC816 Performs a No-Operation.

APPENDIX E

Opcode Matrix

MSD	LSD 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	MSD
0	BRK s 2 8	ORA (d,x) 2 6	COP s 2 ● 8	ORA d,s 2 ★ 4	TSB d 2 ● 5	ORA d 2 3	ASL d 2 5	ORA [d] 2 ● 6	PHP s 1 3	ORA # 2 2	ASL A 1 2	PHD s 1 ★ 4	TSB a 3 ● 6	ORA a 3 4	ASL a 3 6	ORA al 4 ★ 5	0
1	BPL r 2 2	ORA (d),y 2 5	ORA (d) 2 ● 5	ORA (d,s),y 2 ★ 7	TRB d 2 ● 5	ORA d,x 2 4	ASL d,x 2 6	ORA [d],y 2 ● 6	CLC i 1 2	ORA a,y 3 4	INC A 1 ● 2	TCS i 1 ★ 2	TRB a 3 ● 6	ORA a,x 3 4	ASL a,x 3 7	ORA al,x 4 ★ 5	1
2	JSR a 3 6	AND (d,x) 2 6	JSL al 4 ★ 8	AND d,s 2 ★ 4	BIT d 2 3	AND d 2 3	ROL d 2 5	AND [d] 2 ● 6	PLP s 1 4	AND # 2 2	ROL A 1 2	PLD s 1 ★ 5	BIT a 3 4	AND a 3 4	ROL a 3 6	AND al 4 ★ 5	2
3	BMI r 2 2	AND (d),y 2 5	AND (d) 2 ● 5	AND (d,s),y 2 ★ 7	BIT d,x 2 ● 4	AND d,x 2 4	ROL d,x 2 6	AND [d],y 2 ● 6	SEC i 1 2	AND a,y 3 4	DEC A 1 ● 2	TSC i 1 ★ 2	BIT a,x 3 ● 4	AND a,x 3 4	ROL a,x 3 7	AND al,x 4 ★ 5	3
4	RTI s 1 7	EOR (d,x) 2 6	WDM 2 ★ 2	EOR d,s 2 ★ 4	MVP xyc 3 ★ 7	EOR d 2 3	LSR d 2 5	EOR [d] 2 ● 6	PHA s 1 3	EOR # 2 2	LSR A 1 2	PHK s 1 ★ 3	JMP a 3 3	EOR a 3 4	LSR a 3 6	EOR al 4 ★ 5	4
5	BVC r 2 2	EOR (d),y 2 5	EOR (d) 2 ● 5	EOR (d,s),y 2 ★ 7	MVN xyc 3 ★ 7	EOR d,x 2 4	LSR d,x 2 6	EOR [d],y 2 ● 6	CLI i 1 2	EOR a,y 3 4	PHY s 1 ● 3	TCD i 1 ★ 2	JMP al 4 ★ 4	EOR a,x 3 4	LSR a,x 3 7	EOR al,x 4 ★ 5	5
6	RTS s 1 6	ADC (d,x) 2 6	PER s 3 ★ 6	ADC d,s 2 ★ 4	STZ d 2 ● 3	ADC d 2 3	ROR d 2 5	ADC [d] 2 ● 6	PLA s 1 4	ADC # 2 2	ROR A 1 2	RTL s 1 ★ 6	JMP (a) 3 5	ADC a 3 4	ROR a 3 6	ADC al 4 ★ 5	6
7	BVS r 2 2	ADC (d),y 2 5	ADC (d) 2 ● 5	ADC (d,s),y 2 ★ 7	STZ d,x 2 ● 4	ADC d,x 2 4	ROR d,x 2 6	ADC [d],y 2 ● 6	SEI i 1 2	ADC a,y 3 4	PLY s 1 ● 4	TDC i 1 ★ 2	JMP (a,x) 3 ● 6	ADC a,x 3 4	ROR a,x 3 7	ADC al,x 4 ★ 5	7
8	BRA r 2 ● 2	STA (d,x) 2 6	BRL rl 3 ★ 3	STA d,s 2 ★ 4	STY d 2 3	STA d 2 3	STX d 2 3	STA [d] 2 ● 6	DEY i 1 2	BIT # 2 ● 2	TXA i 1 2	PHB s 1 ★ 3	STY a 3 4	STA a 3 4	STX a 3 4	STA al 4 ★ 5	8
9	BCC r 2 2	STA (d),y 2 6	STA (d) 2 ● 5	STA (d,s),y 2 ★ 7	STY d,x 2 4	STA d,x 2 4	STX d,y 2 4	STA [d],y 2 ● 6	TYA i 1 2	STA a,y 3 5	TXS i 1 2	TXY i 1 ★ 2	STZ a 3 ● 4	STA a,x 3 5	STZ a,x 3 ● 5	STA al,x 4 ★ 5	9
A	LDY # 2 2	LDA (d,x) 2 6	LDX # 2 2	LDA d,s 2 ★ 4	LDY d 2 3	LDA d 2 3	LDX d 2 3	LDA [d] 2 ● 6	TAY i 1 2	LDA # 2 2	TAX i 1 2	PLB s 1 ★ 4	LDY a 3 4	LDA a 3 4	LDX a 3 4	LDA al 4 ★ 5	A
B	BCS r 2 2	LDA (d),y 2 5	LDA (d) 2 ● 5	LDA (d,s),y 2 ★ 7	LDY d,x 2 4	LDA d,x 2 4	LDX d,y 2 4	LDA [d],y 2 ● 6	CLV i 1 2	LDA a,y 3 4	TSX i 1 2	TYX i 1 ★ 2	LDY a,x 3 4	LDA a,x 3 4	LDX a,y 3 4	LDA al,x 4 ★ 5	B
C	CPY # 2 2	CMP (d,x) 2 6	REP # 2 3	CMP d,s 2 ★ 4	CPY d 2 3	CMP d 2 3	DEC d 2 5	CMP [d] 2 ● 6	INY i 1 2	CMP # 2 2	DEX i 1 2	WAI i 1 ● 3	CPY a 3 4	CMP a 3 4	DEC a 3 6	CMP al 4 ★ 5	C
D	BNE r 2 2	CMP (d),y 2 5	CMP (d) 2 ● 5	CMP (d,s),y 2 ★ 7	PEI s 2 ● 6	CMP d,x 2 4	DEC d,x 2 6	CMP [d],y 2 ● 6	CLD i 1 2	CMP a,y 3 4	PHX s 1 ● 3	STP i 1 ● 3	JML (a) 3 ★ 6	CMP a,x 3 4	DEC a,x 3 7	CMP al,x 4 ★ 5	D
E	CPX # 2 2	SBC (d,x) 2 6	SEP # 2 3	SBC d,s 2 ★ 4	CPX d 2 3	SBC d 2 3	INC d 2 5	SBC [d] 2 ● 6	INX i 1 2	SBC # 2 2	NOP i 1 2	XBA i 1 ★ 3	CPX a 3 4	SBC a 3 4	INC a 3 6	SBC al 4 ★ 5	E
F	BEQ r 2 2	SBC (d),y 2 5	SBC (d) 2 ● 5	SBC (d,s),y 2 ★ 7	PEA s 3 ★ 5	SBC d,x 2 4	INC d,x 2 6	SBC [d],y 2 ● 6	SED i 1 2	SBC a,y 3 4	PLX s 1 ● 4	XCE i 1 ★ 2	JSR (a,x) 3 ★ 6	SBC a,x 3 4	INC a,x 3 7	SBC al,x 4 ★ 5	F
MSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	MSD

symbol	addressing mode	symbol	addressing mode
#	immediate	[d]	direct indirect long
A	accumulator	[d],y	direct indirect long indexed
r	program counter relative	a	absolute
rl	program counter relative long	a,x	absolute indexed (with x)
i	implied	a,y	absolute indexed (with y)
s	stack	al	absolute long
d	direct	al,x	absolute long indexed
d,x	direct indexed (with x)	d,s	stack relative
d,y	direct indexed (with y)	(d,s),y	stack relative indirect indexed
(d)	direct indirect	(a)	absolute indirect
(d,x)	direct indexed indirect	(a,x)	absolute indexed indirect
(d),y	direct indirect indexed	xyc	block move

Op Code Matrix Legend

INSTRUCTION MNEMONIC

BASE NO. BYTES

★ = New W65C816/802 Opcodes
● = New W65C02 Opcodes
Blank = NMOS 6502 Opcodes

ADDRESSING MODE

BASE NO CYCLES

APPENDIX F

Operation, Operation Codes, and Status Register

MNE-MONIC	OPERATION	#	a	a|i	d	A	i	(d),y	[d],y	(d,x)	d,x	a,x	a|i,x	a,y	r	rf	(a)	(d)	[d]	(a,x)	s	d,s	(d,s),y	xyc	PROCESSOR STATUS CODE 7 6 5 4 3 2 1 0 N V M X D I Z C	MNE-MONIC E 0
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	N V 1 B D I Z C E - 1
ADC	A + M + C → A	69	6D	6F	65			71	77	61	75		7D	7F	79				72	67			63	73		N V . . . . Z C	ADC
AND	A∨M → A	29	2D	2F	25			31	37	21	35		3D	3F	39				32	27			23	33		N . . . . . Z .	AND
ASL	C ← 15 7 0 ← 0		0E		06	0A					16		1E													N . . . . . Z C	ASL
BCC	BRANCH IF C = 0															90											BCC
BCS	BRANCH IF C = 1															B0											BCS
BEQ	BRANCH IF Z = 1															F0											BEQ
BIT	A∨M (NOTE 1)	89	2C		24						34		3C													M M . . . . Z .	BIT
BMI	BRANCH IF N = 1															30											BMI
BNE	BRANCH IF Z = 0															D0											BNE
BPL	BRANCH IF N = 0															10											BPL
BRA	BRANCH ALWAYS															80											* BRA
BRK	BREAK (NOTE 2)																					00				• 0 1	* BRK
BRL	BRANCH LONG ALWAYS																82										* BRL
BVC	BRANCH IF V = 0															50											BVC
BVS	BRANCH IF V = 1															70											BVS
CLC	0 → C						18																			 0	CLC
CLD	0 → D						D8																			. . . 0 . . .	CLD
CLI	0 → I						58																			. . . . 0 . .	CLI
CLV	0 → V						B8																			. 0	CLV
CMP	A - M	C9	CD	CF	C5			D1	D7	C1	D5		DD	DF	D9				D2	C7			C3	D3		N . . . . . Z C	CMP
COP	CO-PROCESSOR																					02					* COP
CPX	X-M	E0	EC		E4																					N . . . . . Z C	CPX
CPY	Y-M	C0	CC		C4																					N . . . . . Z C	CPY
DEC	DECREMENT		CE		C6	3A					D6		DE													N . . . . . Z .	DEC
DEX	X - 1 → X						CA																			N . . . . . Z .	DEX
DEY	Y - 1 → Y						88																			N . . . . . Z .	DEY
EOR	A∨M → A	49	4D	4F	45			51	57	41	55		5D	5F	59				52	47			43	53		N . . . . . Z .	EOR
INC	INCREMENTS		EE		E6	1A					F6		FE													N . . . . . Z .	INC
INX	X + 1 → X						E8																			N . . . . . Z .	INX
INY	Y + 1 → Y						C8																			N . . . . . Z .	INY
JML	JUMP LONG TO NEW LOC																	DC									* JML
JMP	JUMP TO NEW LOC		4C	5C														6C			7C						JMP
JSL	JUMP LONG TO SUB			22																							JSL
JSR	JUMP TO SUB		20																		FC						JSR
LDA	M → A	A9	AD	AF	A5			B1	B7	A1	B5		BD	BF	B9				B2	A7		A3	B3			N . . . . . Z .	LDA
LDX	M → X	A2	AE		A6							B6			BE											N . . . . . Z .	LDX
LDY	M → Y	A0	AC		A4						B4		BC													N . . . . . Z .	LDY
LSR	0 → 15/7 0 → C		4E		46	4A					56		5E													0 . . . . . Z C	LSR
MVN	M → M BACKWARD																								54		* MVN
MVP	M → M FORWARD																								44		* MVP
NOP	NO OPERATION						EA																				NOP
ORA	A∨M → A	09	0D	0F	05			11	17	01	15		1D	1F	19				12	07			03	13		N . . . . . Z .	ORA
PEA	Mpc + 1, Mpc + 2 → Ms, Ms - 1 S - 2 → S																					F4					* PEA
PEI	M(d), M(d + 1) → Ms, Ms - 1 S - 2 → S																					D4					* PEI
PER	Mpc + rl, Mpc + rl + 2 → Ms, MS - 1 S - 2 → S																					62					* PER
PHA	A → Ms, S - 2 → S S - 1 → S																					48					PHA
PHB	DBR → Ms, S - 1 → S																					8B					PHB
PHD	D → Ms, Ms - 1, S - 2 → S																					0B					* PHD
PHK	PBR → Ms, S - 1 → S																					4B					* PHK
PHP	P → Ms, S - 1 → S																					08					PHP
PHX	X → Ms, S - 1 → S																					DA					• PHX
PHY	Y → Ms, S - 1 → S																					5A					• PHY
PLA	S + 1 → S, Ms → A																					68				N . . . . . Z .	PLA
PLB	S + 1 → S, Ms → DBR																					AB				N . . . . . Z .	* PLB
PLD	S + 2 → S, Ms - 1, Ms → D																					2B				N . . . . . Z .	* PLD
PLP	S + 1 → S, Ms → P																					28				N V M X D I Z C	PLP
PLX	S + 1 → S, Ms → X																					FA				N . . . . . Z .	PLX
PLY	S + 1 → S, Ms → Y																					7A				N . . . . . Z .	PLY
REP	MVP → P	C2																								N V M X D I Z C	* REP
ROL	15/7 0 ← C		2E		26	2A					36		3E													N . . . . . Z C	ROL
ROR	C → 15/7 0		6E		66	6A					76		7E													N . . . . . Z C	ROR
RTI	RTRN FROM INT																					40				N V M X D I Z C	RTI
RTL	RTRN FROM SUB, LONG																					6B				. . . . .	* RTL
RTS	RTRN SUBROUTINE																					60				. . . . .	RTS
SBC	A - M - C → A	E9	ED	EF	E5			F1	F7	E1	F5		FD	FF	F9				F2	E7			E3	F3		N V . . . . . Z C	SBC
SEC	1 → C						38																			 1	SEC
SED	1 → D						F8																			. . . . 1 . .	SED
SEI	1 → I						78																			. . . . . 1 .	SEI
SEP	MVP → P	E2																								N V M X D I Z C	* SEP
STA	A → M		8D	8F	85			91	97	81	95		9D	9F	99				92	87			83	93		. . . . .	STA
STP	STOP (1 → ϕ2)						DB																				• STP
STX	X → M		8E		86							96															STX
STY	Y → M		8C		84						94																STY
STZ	00 → M		9C		64						74		9E														• STZ
TAX	A → X						AA																			N . . . . . Z .	TAX
TAY	A → Y						A8																			N . . . . . Z .	TAY
TCD	C → D						5B																			N . . . . . Z .	* TCD
TCS	C → S						1B																			. . . . .	* TCS
TDC	D → C						7B																			N . . . . . Z .	* TDC
TRB			1C		14																					. . . . . Z .	• TRB
TSB	AVM → M		0C		04																					. . . . . Z .	• TSB
TSC	S → C						3B																			N . . . . . Z .	* TSC
TSX	S → X						BA																			N . . . . . Z .	TSX
TXA	X → A						8A																			N . . . . . Z .	TXA
TXS	X → S						9A																			. . . . .	TXS
TXY	X → Y						9B																			N . . . . . Z .	* TXY
TYA	Y → A						98																			N . . . . . Z .	* TYA
TYX	Y → X						BB																			N . . . . . Z .	• TYX
WAI	D → RDY						CB																			. . . . .	• WAI
WDM	NO OPERATION (RESERVED)						42																			. . . . .	• WDM
XBA	B ⟷ A						EB																			N . . . . . Z .	* XBA
XCE	C ⟷ E						FB																			. . . . . E	* XCE

Notes
1 Bit immediate N and V flags not affected When M = 0, M_{15} = N and M_{14} = V
2 Break Bit (B) in Status register indicates hardware or software break

3 * = New W65C816/802 Instructions
• = New W65C02 Instructions
Blank = NMOS 6502

APPENDIX G

Detailed Instruction Operation

	ADDRESS MODE		CYCLE	VP	ML	VDA	VPA	ADDRESS BUS	DATA BUS	R/W
1	Immediate #		1	1	1	1	1	PBR,PC	Op Code	1
	(LDY,CPY,CPX,LDX,ORA		2	1	1	0	1	PBR,PC+1	IDL	1
	AND,EOR,ADC,BIT,LDA,	(1)(8)	2a	1	1	0	1	PBR,PC+2	IDH	1
	CMP,SBC,REP,SEP)									
	(14 Op Codes)									
	(2 and 3 bytes)									
	(2 and 3 cycles)									
2a	Absolute a		1	1	1	1	1	PBR,PC	Op Code	1
	(BIT,STY,STZ,LDY,		2	1	1	0	1	PBR,PC+1	AAL	1
	CPY,CPX,STX,LDX		3	1	1	0	1	PBR,PC+2	AAH	1
	ORA,AND,EOR,ADC,		4	1	1	1	0	DBR,AA	Data Low	1/0
	STA,LDA,CMP,SBC)	(1)	4a	1	1	1	0	DBR,AA+1	Data High	1/0
	(16 Op Codes)									
	(3 bytes)									
	(4 and 5 cycles									
2b	Absolute (R-M-W) a		1	1	1	1	1	PBR,PC	Op Code	1
			2	1	1	0	1	PBR,PC+1	AAL	1
	(ASL,ROL,LSR,ROR		3	1	1	0	1	PBR,PC+2	AAH	1
	DEC,INC,TSB,TRB)		4	1	0	1	0	DBR,AA	Data Low	1
	(6 Op Codes)	(1)	4a	1	0	1	0	DBR,AA+1	Data High	1
	(3 bytes)		5	1	0	0	0	DBR,AA+1	IO	1
	(6 and 8 cycles)	(3)(1)	6a	1	0	1	0	DBR,AA+1	Data High	0
			6	1	0	1	0	DBR,AA	Data Low	0
2c	Absolute (JUMP) a		1	1	1	1	1	PBR,PC	Op Code	1
	(JMP)(4C)		2	1	1	0	1	PBR,PC+1	NEW PCL	1
	(1 Op Code)		3	1	1	0	1	PBR,PC+2	NEW PCH	1
	(3 bytes)		1	1	1	1	1	PBR, NEW PC	Op Code	1
	(3 cycles)									
2d	Absolute (Jump to		1	1	1	1	1	PBR,PC	Op Code	1
	subroutine) a		2	1	1	0	1	PBR,PC+1	NEW PCL	1
	(JSR)		3	1	1	0	1	PBR,PC+2	NEW PCH	1
	(1 Op Code)		4	1	1	0	0	PBR,PC+2	IO	1
	(3 bytes)		5	1	1	1	0	0,S	PCH	0
	(6 cycles)		6	1	1	1	0	0,S-1	PCL	0
	(different order from M6502)		1	1	1	1	1	PBR NEW PC	Next Op Code	1
*3a	Absolute Long al		1	1	1	1	1	PBR,PC	Op Code	1
	(ORA,AND,EOR,ADC		2	1	1	0	1	PBR,PC+1	AAL	1
	STA,LDA,CMP,SBC)		3	1	1	0	1	PBR,PC+2	AAH	1
	(8 Op Codes)		4	1	1	0	1	PBR,PC+3	AAB	1
	(4 bytes)		5	1	1	1	0	AAB,AA	Data Low	1/0
	(5 and 6 cycles)	(1)	5a	1	1	1	0	AAB,AA+1	Data High	1/0
*3b	Absolute Long (JUMP) al		1	1	1	1	1	PBR,PC	Op Code	1
	(JMP)		2	1	1	0	1	PBR,PC+1	NEW PCL	1
	(1 Op Code)		3	1	1	0	1	PBR,PC+2	NEW PCH	1
	(4 bytes)		4	1	1	0	1	PBR,PC+3	NEW BR	1
	(4 cycles)		1	1	1	1	1	NEW BR	Op Code	1
								NEW PC		
*3c	Absolute Long (Jump to		1	1	1	1	1	PBR,PC	Op Code	1
	Subroutine Long) al		2	1	1	0	1	PBR,PC+1	NEW PCL	1
	(JSL)		3	1	1	0	1	PBR,PC+2	NEW PCH	1
	(1 Op Code)		4	1	1	1	0	0,S	PBR	0
	(4 bytes)		5	1	1	0	0	0,S	IO	1
	(7 cycles)		6	1	1	0	1	PBR,PC+3	NEW PBR	1
			7	1	1	1	0	0,S-1	PCH	0
			8	1	1	1	0	0,S-2	PCL	0
			1	1	1	1	1	NEW PBR,PC	Next Op Code	1
4a	Direct d		1	1	1	1	1	PBR,PC	Op Code	1
	(BIT,STZ,STY,LDY,		2	1	1	0	1	PBR,PC+1	DO	1
	CPY,CPX,STX,LDX,	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	ORA,AND,EOR,ADC,		3	1	1	1	0	0,D+DO	Data Low	1/0
	STA,LDA,CMP,SBC)	(1)	3a	1	1	1	0	0,D+DO+1	Data High	1/0
	(18 Op Codes)									
	(2 bytes)									
	(3 4 and 5 cycles)									
4b	Direct (R-M-W) d		1	1	1	1	1	PBR,PC	Op Code	1
	(ASL,ROL,LSR,ROR		2	1	1	0	1	PBR,PC+1	DO	1
	DEC,INC,TSB,TRB)	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	(6 Op Codes)		3	1	0	1	0	0,D+DO	Data Low	1
	(2 bytes)	(1)	3a	1	0	1	0	0,D+DO+1	Data High	1
	(5,6,7 and 8 cycles)		4	1	0	0	0	0,D+DO+1	IO	1
		(3)(1)	5a	1	0	1	0	0,D+DO+1	Data High	0
			5	1	0	1	0	0,D+DO	Data Low	0
5	Accumulator A		1	1	1	1	1	PBR,PC	Op Code	1
	(ASL,INC,ROL,DEC,LSR,ROR)		2	1	1	0	0	PBR,PC+1	IO	1
	(6 Op Codes)									
	(1 byte)									
	(2 cycles)									
6a	Implied i		1	1	1	1	1	PBR,PC	Op Code	1
	(DEY, INY, INX, DEX, NOP,		2	1	1	0	0	PBR,PC+1	IO	1
	XCE, TYA, TAY,TXA, TXS,									
	TAX,TSX,TCS,TSC,TCD,									
	TDC,TAY,TYX,CLC,SEC,									
	CLI,SEI,CLV,CLD,SED)									
	(25 Op Codes)									
	(1 byte)									
	(2 cycles)									
*6b	Implied i		1	1	1	1	1	PBR,PC	Op Code	1
	(XBA)		2	1	1	0	0	PBR,PC+1	IO	1
	(1 Op Code)		3	1	1	0	0	PBR,PC+1	IO	1
	(1 byte)									
	(3 cycles)									
								RDY		
• 6c	Wait For Interrupt									
	(WAI)		1	1	1	1	1	PBR,PC	Op Code	1
	(1 Op Code)	(9)	2	1	1	0	0	PBR,PC+1	IO	1
	(1 byte)		3	1	1	0	0	PBR,PC+1	IO	1
	(3 cycles)	IRQ NMI	1	1	1	1	1	PBR,PC+1	IRQ(BRK)	1
• 6d	Stop-The-Clock									
	(STP)		1	1	1	1	1	PBR,PC	Op Code	1
	(1 Op Code)		2	1	1	0	0	PBR,PC+1	IO	1
	(1 byte)	RES 1	3	1	1	0	0	PBR,PC+1	IO	1
	(3 cycles)	RES 0	1c	1	1	0	0	PBR,PC+1	RES(BRK)	1
		RES 0	1b	1	1	0	0	PBR,PC+1	RES(BRK)	1
		RES 1	1a	1	1	0	0	PBR,PC+1	RES(BRK)	1
	See 21a Stack (Hardware Interrupt)		1	1	1	1	1	PBR,PC+1	BEGIN	1

	ADDRESS MODE		CYCLE	VP	ML	VDA	VPA	ADDRESS BUS	DATA BUS	R/W
7	Direct Indirect Indexed (d),y		1	1	1	1	1	PBR,PC	Op Code	1
	(ORA,AND,EOR,ADC,		2	1	1	0	1	PBR,PC+1	DO	1
	STA,LDA,CMP,SBC)	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	(8 Op Codes)		3	1	1	1	0	0,D+DO	AAL	1
	(2 bytes)		4	1	1	1	0	0,D+DO+1	AAH	1
	(5,6,7 and 8 cycles)	(4)	4a	1	1	0	0	0,D+DO+1	IO	1
			5	1	1	1	0	DBR,AA+Y	Data Low	1/0
		(1)	5a	1	1	1	0	DBR,AA+Y+1	Data High	1/0
8	Direct Indirect		1	1	1	1	1	PBR,PC	Op Code	1
	Indexed Long [d],y		2	1	1	0	1	PBR,PC+1	DO	1
	(ORA,AND,EOR,ADC	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	STA,LDA,CMP,SBC)		3	1	1	1	0	0,D+DO	AAL	1
	(8 Op Codes)		4	1	1	1	0	0,D+DO+1	AAH	1
	(2 bytes)		5	1	1	1	0	0,D+DO+2	AAB	1
	(6,7 and 8 cycles)		6	1	1	1	0	AAB,AA+Y	Data Low	1/0
		(1)	6a	1	1	1	0	AAB,AA+Y+1	Data High	1/0
9	Direct Indexed Indirect (d,x)		1	1	1	1	1	PBR,PC	Op Code	1
	(ORA,AND,EOR,ADC		2	1	1	0	1	PBR,PC+1	DO	1
	STA,LDA,CMP,SBC)	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	(8 Op Codes)		3	1	1	0	0	PBR,PC+1	IO	1
	(2 bytes)		4	1	1	1	0	0,D+DO+X	AAL	1
	(6,7 and 8 cycles)		5	1	1	1	0	0,D+DO+X+1	AAH	1
			6	1	1	1	0	DBR,AA	Data Low	1/0
		(1)	6a	1	1	1	0	DBR,AA+1	Data High	1/0
10a	Direct X d,x		1	1	1	1	1	PBR,PC	Op Code	1
	(BIT,STZ,STY,LDY		2	1	1	0	1	PBR,PC+1	DO	1
	ORA,AND,EOR,ADC	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	STA,LDA,CMP,SBC)		3	1	1	0	0	PBR,PC+1	IO	1
	(11 Op Codes)		4	1	1	1	0	0,D+DO+X	Data Low	1/0
	(2 bytes)	(1)	4a	1	1	1	0	0,D+DO+X+1	Data High	1/0
	(4,5 and 6 cycles)									
10b	Direct,X(R-M-W) d,x		1	1	1	1	1	PBR,PC	Op Code	1
	(ASL,ROL,LSR,ROR		2	1	1	0	1	PBR,PC+1	DO	1
	DEC,INC)	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	(6 Op Codes)		3	1	1	0	0	PBR,PC+1	IO	1
	(2 bytes)		4	1	0	1	0	0,D+DO+X	Data Low	1
	(6,7,8 and 9 cycles)	(1)	4a	1	0	1	0	0,D+DO+X+1	Data High	1
			5	1	0	0	0	0,D+DO+X+1	IO	1
		(3)(1)	6a	1	0	1	0	0,D+DO+X+1	Data High	0
			6	1	0	1	0	0,D+DO+X	Data Low	0
11	Direct,Y d,y		1	1	1	1	1	PBR,PC	Op Code	1
	(STX,LDX)		2	1	1	0	1	PBR,PC+1	DO	1
	(2 Op Codes)	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	(2 bytes)		3	1	1	0	0	PBR,PC+1	IO	1
	(4,5 and 6 cycles)		4	1	1	1	0	0,D+DO+Y	Data Low	1/0
		(1)	4a	1	1	1	0	0,D+DO+Y+1	Data High	1/0
12a	Absolute,X a,x		1	1	1	1	1	PBR,PC	Op Code	1
	(BIT,LDY,STZ,		2	1	1	0	1	PBR,PC+1	AAL	1
	ORA,AND,EOR,ADC,		3	1	1	0	1	PBR,PC+2	AAH	1
	STA,LDA,CMP,SBC)		3a	1	1	0	0	PBR,PC+2	IO	1
	(11 Op Codes)		4	1	1	1	0	DBR,AA+X	Data Low	1/0
	(3 bytes)	(1)	4a	1	1	1	0	DBR,AA+X+1	Data High	1/0
	(4,5 and 6 cycles)									
12b	Absolute,X(R-M-W) a,x		1	1	1	1	1	PBR,PC	Op Code	1
	(ASL,ROL,LSR,ROR,		2	1	1	0	1	PBR,PC+1	AAL	1
	DEC,INC)		3	1	1	0	1	PBR,PC+2	AAH	1
	(6 Op Codes)	(4)	3a	1	1	0	0	PBR,PC+2	IO	1
	(3 bytes)		4	1	0	1	0	DBR,AA+X	Data Low	1
	(6,7,8 and 9 cycles)	(1)	4a	1	0	1	0	DBR,AA+X+1	Data High	1
			5	1	0	0	0	DBR,AA+X+1	IO	1
		(3)(1)	6a	1	0	1	0	DBR,AA+X+1	Data High	0
			6	1	0	1	0	DBR,AA+X	Data Low	0
*13	Absolute Long,X al,x		1	1	1	1	1	PBR,PC	Op Code	1
	(ORA,AND,EOR,ADC		2	1	1	0	1	PBR,PC+1	AAL	1
	STA,LDA,CMP,SBC)		3	1	1	0	1	PBR,PC+2	AAH	1
	(8 Op Codes)		4	1	1	0	1	PBR,PC+3	AAB	1
	(4 bytes)		5	1	1	1	0	AAB,AA+X	Data Low	1/0
	(5 and 6 cycles)	(1)	5a	1	1	1	0	AAB,AA+X+1	Data High	1/0
14	Absolute Y a,y		1	1	1	1	1	PBR,PC	Op Code	1
	(LDX,ORA,AND,EOR,ADC		2	1	1	0	1	PBR,PC+1	AAL	1
	STA,LDA,CMP,SBC)		3	1	1	0	1	PBR,PC+2	AAH	1
	(9 Op Codes)	(4)	3a	1	1	0	0	PBR,PC+2	IO	1
	(3 bytes)		4	1	1	1	0	DBR,AA+Y	Data Low	1/0
	(4,5 and 6 cycles)	(1)	4a	1	1	1	0	DBR,AA+Y+1	Data High	1/0
15	Relative r		1	1	1	1	1	PBR,PC	Op Code	1
	(BPL,BMI,BVC,BVS,BCC		2	1	1	0	1	PBR,PC+1	Offset	1
	BCS,BNE,BEQ,BRA)	(5)	2a	1	1	0	0	PBR,PC+1	IO	1
	(9 Op Codes)	(6)	2b	1	1	0	0	PBR,PC+1	IO	1
	(2 bytes)		1	1	1	1	1	PBR,PC+Offset	Op Code	1
	(2 3 and 4 cycles)									
*16	Relative Long rl		1	1	1	1	1	PBR,PC	Op Code	1
	(BRL)		2	1	1	0	1	PBR,PC+1	Offset Low	1
	(1 Op Code)		3	1	1	0	1	PBR,PC+2	Offset High	1
	(3 bytes)		4	1	1	0	0	PBR,PC+2	IO	1
	(4 cycles)		1	1	1	1	1	PBR,PC+Offset	Op Code	1
17a	Absolute Indirect (a)		1	1	1	1	1	PBR,PC	Op Code	1
	(JMP)		2	1	1	0	1	PBR,PC+1	AAL	1
	(1 Op Code)		3	1	1	0	1	PBR,PC+2	AAH	1
	(3 bytes)		4	1	1	1	0	0,AA	NEW PCL	1
	(5 cycles)		5	1	1	1	0	0,AA+1	NEW PCH	1
			1	1	1	1	1	NEW,PBR,PC	Op Code	1
*17b	Absolute Indirect (a)		1	1	1	1	1	PBR,PC	Op Code	1
			2	1	1	0	1	PBR,PC+1	AAL	1
	(JML)		3	1	1	0	1	PBR,PC+2	AAH	1
	(1 Op Code)		4	1	1	1	0	0,AA	NEW PCL	1
	(3 bytes)		5	1	1	1	0	0,AA+1	NEW PCH	1
	(6 cycles)		6	1	1	1	0	0,AA+2	NEW PBR	1
			1	1	1	1	1	NEW,PBR,PC	Op Code	1
• 18	Direct Indirect (d)		1	1	1	1	1	PBR,PC	Op Code	1
	(ORA,AND,EOR,ADC		2	1	1	0	1	PBR,PC+1	DO	1
	STA,LDA,CMP,SBC)	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	(8 Op Codes)		3	1	1	1	0	0,D+DO	AAL	1
	(2 bytes)		4	1	1	1	0	0,D+DO+1	AAH	1
	(5,6 and 7 cycles)		5	1	1	1	0	DBR,AA	Data Low	1/0
		(1)	5a	1	1	1	0	DBR,AA+1	Data Low	1/0

APPENDIX G

Detailed Instruction Operation (continued)

	ADDRESS MODE		CYCLE	VP	ML	VDA	VPA	ADDRESS BUS	DATA BUS	R/W
*19	Direct Indirect Long [d]		1	1	1	1	1	PBR PC	Op Code	1
	(ORA,AND,EOR,ADC		2	1	1	0	1	PBR PC+1	DO	1
	STA,LDA,CMP,SBC)	(2)	2a	1	1	0	0	PBR PC+1	IO	1
	(8 Op Codes)		3	1	1	1	0	0,D+DO	AAL	1
	(2 bytes)		4	1	1	1	0	0,D+DO+1	AAH	1
	(6,7 and 8 cycles)		5	1	1	1	0	0,D+DO+2	AAB	1
			6	1	1	1	0	AAB,AA	Data Low	1/0
		(1)	6a	1	1	1	0	AAB,AA+1	Data High	1/0
20a	Absolute Indexed Indirect (a,x)		1	1	1	1	1	PBR PC	Op Code	1
	(JMP)		2	1	1	0	1	PBR PC+1	AAL	1
	(1 Op Code)		3	1	1	0	1	PBR PC+2	AAH	1
	(3 bytes)		4	1	1	0	0	PBR PC+2	IO	1
	(6 cycles)		5	1	1	0	1	PBR AA+X	NEW PCL	1
			6	1	1	0	1	PBR AA+X+1	NEW PCH	1
			1	1	1	1	1	PBR NEW PC	Op Code	1
*20b	Absolute Indexed Indirect (Jump to Subroutine Indexed Indirect) (a,x) (JSR)		1	1	1	1	1	PBR PC	Op Code	1
			2	1	1	0	1	PBR PC+1	AAL	1
			3	1	1	1	0	0,S	PCH	0
			4	1	1	1	0	0,S-1	PCL	0
	(1 Op Code)		5	1	1	0	1	PBR PC+2	AAH	1
	(3 bytes)		6	1	1	0	0	PBR PC+2	IO	1
	(8 cycles)		7	1	1	0	1	PBR AA+X	NEW PCL	1
			8	1	1	0	1	PBR AA+X+1	NEW PCH	1
			1	1	1	1	1	PBR,NEW PC	Next Op Code	1
21a	Stack (Hardware Interrupts) s		1	1	1	1	1	PBR PC	IO	1
			2	1	1	0	0	PBR PC	IO	1
	(IRQ,NMI,ABORT,RES)	(7)	3	1	1	1	0	0,S	PBR	0
	(4 hardware interrupts)		4	1	1	1	0	0,S-1	PCH	0
	(0 bytes)		5	1	1	1	0	0,S-2	PCL	0
	(7 and 8 cycles)		6	1	1	1	0	0,S-3	P	0
			7	0	1	1	0	0,VA	AAVL	1
			8	0	1	1	0	0,VA+1	AAVH	1
			1	1	1	1	1	0,AAV	Next Op Code	1
21b	Stack (Software Interrupts) s		1	1	1	1	1	PBR PC	Op Code	1
			2	1	1	0	1	PBR PC+1	Signature	1
	(BRK,COP)	(7)	3	1	1	1	0	0,S	PBR	0
	(2 Op Codes)		4	1	1	1	0	0,S-1	PCH	0
	(2 bytes)		5	1	1	1	0	0,S-2	PCL	0
	(7 and 8 cycles)		6	1	1	1	0	0,S-3 (COP Latches)	P	0
			7	0	1	1	0	0,VA	AAVL	1
			8	0	1	1	0	0,VA+1	AAVH	1
			1	1	1	1	1	0,AAV	Next Op Code	1
21c	Stack (Return from Interrupt) s		1	1	1	1	1	PBR PC	Op Code	1
			2	1	1	0	0	PBR PC+1	IO	1
	(RTI)		3	1	1	0	0	PBR PC+1	IO	1
	(1 Op Code)		4	1	1	1	0	0,S+1	P	1
	(1 byte)		5	1	1	1	0	0,S+2	PCL	1
	(6 and 7 cycles)		6	1	1	1	0	0,S+3	PCH	1
	(different order from N6502)	(7)	7	1	1	1	0	0,S+4	PBR	1
			1	1	1	1	1	PBR PC	New Op Code	1
21d	Stack (Return from Subroutine) s		1	1	1	1	1	PBR PC	Op Code	1
			2	1	1	0	0	PBR PC+1	IO	1
	(RTS)		3	1	1	0	0	PBR PC+1	IO	1
	(1 Op Code)		4	1	1	1	0	0,S+1	PCL	1
	(1 byte)		5	1	1	1	0	0,S+2	PCH	1
	(6 cycles)		6	1	1	1	0	0,S+3	IO	1
			1	1	1	1	1	PBR PC	Op Code	1
*21e	Stack (Return from Subroutine Long) s		1	1	1	1	1	PBR PC	Op Code	1
			2	1	1	0	0	PBR PC+1	IO	1
	(RTL)		3	1	1	0	0	PBR PC+1	IO	1
	(1 Op Code)		4	1	1	1	0	0,S+1	NEW PCL	1
	(1 byte)		5	1	1	1	0	0,S+2	NEW PCH	1
	(6 cycles)		6	1	1	1	0	0,S+3	NEW PBR	1
			1	1	1	1	1	NEW PBR PC	Next Op Code	1
21f	Stack (Push) s		1	1	1	1	1	PBR PC	Op Code	1
	(PHP,PHA,PHY,PHX,		2	1	1	0	0	PBR PC+1	IO	1
	PHD,PHK,PHB)	(1)	3a	1	1	1	0	0,S	Register High	1
	(7 Op Codes)		3	1	1	1	0	0,S-1	Register Low	1
	(1 byte)									
	(3 and 4 cycles)									
21g	Stack (Pull) s		1	1	1	1	1	PBR PC	Op Code	1
	(PLP,PLA,PLY,PLX,PLD,PLB)		2	1	1	0	0	PBR PC+1	IO	1
	(Different than N6502)		3	1	1	0	0	PBR PC+1	IO	1
	(6 Op Codes)		4	1	1	1	0	0,S+1	Register Low	1
	(1 byte)	(1)	4a	1	1	1	0	0,S+2	Register High	1
	(4 and 5 cycles)									
*21h	Stack (Push Effective Indirect Address) s		1	1	1	1	1	PBR PC	Op Code	1
			2	1	1	0	1	PBR PC+1	DO	1
	(PEI)	(2)	2a	1	1	0	0	PBR PC+1	IO	1
	(1 Op Code)		3	1	1	1	0	0,D+DO	AAL	1
	(2 bytes)		4	1	1	1	0	0,D+DO+1	AAH	1
	(6 and 7 cycles)		5	1	1	1	0	0,S	AAH	0
			6	1	1	1	0	0,S-1	AAL	0
*21i	Stack (Push Effective Absolute Address) s		1	1	1	1	1	PBR PC	Op Code	1
			2	1	1	0	1	PBR PC+1	AAL	1
	(PEA)		3	1	1	0	1	PBR PC+2	AAH	1
	(1 Op Code)		4	1	1	1	0	0,S	AAH	0
	(3 bytes)		5	1	1	1	0	0,S-1	AAL	0
	(5 cycles)									
*21j	Stack (Push Effective Program Counter Relative Address) s		1	1	1	1	1	PBR PC	Op Code	1
			2	1	1	0	1	PBR PC+1	Offset Low	1
			3	1	1	0	1	PBR PC+2	Offset High	1
	(PER)		4	1	1	0	0	PBR PC+2	IO	1
	(1 Op Code)		5	1	1	1	0	0,S	PCH+OFF+CARRY	0
	(3 bytes)									
	(6 cycles)		6	1	1	1	0	0,S-1	PCL+OFFSET	0
*22	Stack Relative d,s		1	1	1	1	1	PBR PC	Op Code	1
	(ORA,AND,EOR,ADC,		2	1	1	0	1	PBR PC+1	SO	1
	STA,LDA,CMP,SBC)		3	1	1	0	0	PBR PC+1	IO	1
	(8 Op Codes)		4	1	1	1	0	0,S+SO	Data Low	1/0
	(2 bytes)	(1)	4a	1	1	1	0	0,S+SO+1	Data High	1/0
	(4 and 5 cycles)									

	ADDRESS MODE		CYCLE	VP	ML	VDA	VPA	ADDRESS BUS	DATA BUS	R/W
*23	Stack Relative Indirect Indexed (d,s),y (ORA,AND,EOR,ADC, STA,LDA,CMP,SBC) (8 Op Codes) (2 bytes) (7 and 8 Cycles)		1	1	1	1	1	PBR PC	Op Code	1
			2	1	1	0	1	PBR PC+1	SO	1
			3	1	1	0	0	PBR PC+1	IO	1
			4	1	1	1	0	0,S+SO	AAL	1
			5	1	1	1	0	0,S+SO+1	AAH	1
			6	1	1	0	0	DBR AA+Y	Data Low	1/0
		(1)	7a	1	1	1	0	DBR AA+Y+1	Data High	1/0
*24a	Block Move Positive (forward) xyc (MVP) (1 Op Code) (3 bytes) (7 cycles) a = Source Address y = Destination c = Number of Bytes to Move -1 x,y Decrement	N-2 Byte C-2	1	1	1	1	1	PBR PC	Op Code	1
			2	1	1	0	1	PBR PC+1	DBA	1
			3	1	1	0	1	PBR PC+2	SBA	1
			4	1	1	1	0	SBA,X	Source Data	1
			5	1	1	1	0	DBA,Y	Dest Data	0
			6	1	1	0	0	DBA,Y	IO	1
			7	1	1	0	0	DBA,Y	IO	1
	FFFFFF / Dest Start / Source Start / Dest End / Source End / 000000	N Byte Last C+0	1	1	1	1	1	PBR PC	Op Code	1
			2	1	1	0	1	PBR PC+1	DBA	1
			3	1	1	0	1	PBR PC+2	SBA	1
			4	1	1	1	0	SBA,X-2	Source Data	1
			5	1	1	1	0	DBA,Y-2	Dest Data	0
			6	1	1	0	0	DBA,Y-2	IO	1
			7	1	1	0	0	DBA,Y-2	IO	1
			1	1	1	1	1	PBR,PC-3	Next Op Code	1
*24b	Block Move Negative (backward) xyc (MVN) (1 Op Code) (3 bytes) (7 cycles) a = Source Address y = Destination c = Number of Bytes to Move -1 x,y Increment	N-2 Byte C-2	1	1	1	1	1	PBR PC	Op Code	1
			2	1	1	0	1	PBR PC+1	DBA	1
			3	1	1	0	1	PBR PC+2	SBA	1
			4	1	1	1	0	SBA,X	Source Data	1
			5	1	1	1	0	DBA,Y	Dest Data	0
			6	1	1	0	0	DBA,Y	IO	1
			7	1	1	0	0	DBA,Y	IO	1
	FFFFFF / Source End / Dest End / Source Start / Dest Start / 000000	N-1 Byte C+0	1	1	1	1	1	PBR PC	Op Code	1
			2	1	1	0	1	PBR PC+1	DBA	1
			3	1	1	0	1	PBR PC+2	SBA	1
			4	1	1	1	0	SBA,X+1	Source Data	1
			5	1	1	1	0	DBA,Y+1	Dest Data	0
			6	1	1	0	0	DBA,Y+1	IO	1
			7	1	1	0	0	DBA,Y+1	IO	1
			1	1	1	1	1	PBR PC	Op Code	1
			2	1	1	0	1	PBR PC+1	DBA	1
		N Byte C+0	3	1	1	0	1	PBR PC+2	SBA	1
			4	1	1	1	0	SBA,X+2	Source Data	1
			5	1	1	1	0	DBA,Y+2	Dest Data	0
			6	1	1	0	0	DBA,Y+2	IO	1
			7	1	1	0	0	DBA,Y+2	IO	1
			1	1	1	1	1	PBR PC-3	Next Op Code	1

Notes

(1) Add 1 byte (for immediate only) for M=0 or X=0 (i.e. 16 bit data), add 1 cycle for M=0 or X=0

(2) Add 1 cycle for direct register low (DL) not equal 0

(3) Special case for aborting instruction. This is the last cycle which may be aborted or the status register will be updated. Read-modify-write instructions are the only special cases

(4) Add 1 cycle for indexing across page boundaries, or write, or x=0

(5) Add 1 cycle if branch is taken

(6) Add 1 cycle if branch is taken across page boundaries in 6502 emulation mode (E=1)

(7) Subtract 1 cycle for 6502 emulation mode (E=1)

(8) Add 1 cycle for REP,SEP

(9) Wait at cycle 2 for 2 cycles after $\overline{NMI}$ or $\overline{IRQ}$ active input

Abbreviations

AAB Absolute Address Bank
AAH Absolute Address High
AAL Absolute Address Low
AAVH Absolute Address Vector High
AAVL Absolute Address Vector Low
C Accumulator
D Direct Register
DBA Destination Bank Address
DBR Data Bank Register
DO Direct Offset
IDH Immediate Data High
IDL Immediate Data Low
IO Internal Operation
P Status Register
PBR Program Bank Register
PC Program Counter
R-M-W Read-Modify-Write
S Stack Address
SBA Source Bank Address
SO Stack Offset
VA Vector Address
x,y Index Registers
* = New W65C816/802 Addressing Modes
@ = New W65C02 Addressing Modes
Blank = NMOS 6502 Addressing Modes

APPENDIX H

W65C816 Compatibility Issues

	W65C816/802	W65C02	NMOS 6502
1. S (Stack)	Always page 1 (E = 1), 8 bits 16 bits when (E = 0).	Always page 1, 8 bits	Always page 1, 8 bits
2. X (X Index Register)	Indexed page zero always in page 0 (E = 1). Cross page (E = 0)	Always page 0	Always page 0
3. Y (Y Index Register)	Indexed page zero always in page 0 (E = 1). Cross page (E = 0)	Always page 0	Always page 0
4. A (Accumulator)	8 bits (M = 1), 16 bits (M = 0)	8 bits	8 bits
5. P (Flag Register)	N, V, and Z flags valid in decimal mode D = 0 after reset or interrupt.	N, V, and Z flags valid in decimal mode D = 0 after reset and interrupt	N, V, and Z flags invalid in decimal mode. D = unknown after reset. D not modified after interrupt.
6. Timing A. ABS, X ASL, LSR, ROL, ROR With No Page Crossing	7 cycles	6 cycles	7 cycles
B. Jump Indirect Operand = XXFF	5 cycles	6 cycles	5 cycles and invalid page crossing
C. Branch Across Page	4 cycles (E = 1) 3 cycles (E = 0)	4 cycles	4 cycles
D. Decimal Mode	No additional cycle	Add 1 cycle	No additional cycles
7. BRK Vector	00FFFE,F (E = 1) BRK bit = 0 on stack if $\overline{IRQ}$, NMI, $\overline{ABORT}$. 00FFE6, 7 (E = 0) X = X on Stack always.	FFFE,F BRK bit = 0 on stack if $\overline{IRQ}$, $\overline{NMI}$.	FFFE,F BRK bit = 0 on stack if $\overline{IRQ}$, NMI.
8. Interrupt or Break Bank Address	PBR not pushed (E = 1) RTI PBR not pulled (E = 1) PBR pushed (E = 0) RTI PBR pulled (E = 0)	Not available	Not available
9. Memory Lock ($\overline{ML}$)	$\overline{ML}$ = 0 during Read, Modify and Write cycles.	$\overline{ML}$ = 0 during Modify and Write.	Not available
10. Indexed Across Page Boundary	Extra read of last instruction fetch. (Note 1)	Extra read of last instruction fetch.	Extra read of invalid address.
11. RDY Pulled During Write Cycle.	Ignored (E = 1) for W65C802 only. Processor stops (E = 0).	Processor stops	Ignored
12. WAI and STP Instructions.	Available	Available	Not available
13. Unused OP Codes	One reserved OP Code specified as WDM will be used in future systems. The W65C816 performs a no-operation.	No operation	Unknown and some "hang up" processor.
14. Bank Address Handling	PBR = 00 after reset or interrupts.	Not available	Not available
15. R/$\overline{W}$ During Read-Modify-Write Instructions	E = 1, R/$\overline{W}$ = 0 during Modify and Write cycles. E = 0, R/$\overline{W}$ = 0 only during Write cycle.	R/$\overline{W}$ = 0 only during Write cycle	R/$\overline{W}$ = 0 during Modify and Write cycles.
16. Pin 7	W65C802 = SYNC. W65C816 = VPA	SYNC	SYNC
17. COP Instruction Signatures X0-X7, user defined Signatures X8-XF reserved	Available	Not available	Not available

Note 1. Extra read of Invalid Address for STA in following addressing modes: (d),y; a,x; a,y.

Caveats:

These caveats concern wrapping and other unnecessary system configurations.

1. When in the Emulation mode, wrapping stack and page zero for new instructions is not supported.
2. When running programs in the Emulation mode, other than bank zero, during the handling of interrupts, the bank is not pushed onto the stack.
3. Future versions of the W65Cxyz family will not support program bank wrapping or any other form of bank wrapping.

APPENDIX I

W65C802 and W65C816 Microprocessor Addressing Modes

The W65C816 is capable of directly addressing 16 MBytes of memory. This address space has special significance within certain addressing modes, as follows:

Reset and Interrupt Vectors

The Reset and Interrupt vectors use the majority of the fixed addresses between 00FFE0 and 00FFFF.

Stack

The Stack may use memory from 000000 to 00FFFF. The effective address of Stack and Stack Relative addressing modes will always be within this range.

Direct

The Direct addressing modes are usually used to store memory registers and pointers. The effective address generated by Direct, Direct,X and Direct,Y addressing modes is always in Bank 0 (000000-00FFFF).

Program Address Space

The Program Bank register is not affected by the Relative, Relative Long, Absolute, Absolute Indirect, and Absolute Indexed Indirect addressing modes or by incrementing the Program Counter from FFFF. The only instructions that affect the Program Bank register are: RTI, RTL, JML, JSL, and JMP Absolute Long. Program code may exceed 64K bytes although code segments may not span bank boundaries.

Data Address Space

The data address space is contiguous throughout the 16 MByte address space. Words, arrays, records, or any data structures may span 64 KByte bank boundaries with no compromise in code efficiency. The following addressing modes generate 24-bit effective addresses:

- Direct Indexed Indirect (d,x)
- Direct Indirect Indexed (d),y
- Direct Indirect (d)
- Direct Indirect Long [d]
- Direct Indirect Long Indexed [d],y
- Absolute a
- Absolute a,x
- Absolute a,y
- Absolute Long al
- Absolute Long Indexed al,x
- Stack Relative Indirect Indexed (d),y

The following addressing mode descriptions provide additional detail as to how effective addresses are calculated.

Twenty-four addressing modes are available for use with the W65C802 and W65C816 microprocessors. The "long" addressing modes may be used with the W65C802; however, the high byte of the address is not available to the hardware. Detailed descriptions of the 24 addressing modes are as follows:

1. Immediate Addressing—#

The operand is the second byte (second and third bytes when in the 16-bit mode) of the instruction.

2. Absolute—a

With Absolute addressing the second and third bytes of the instruction form the low-order 16 bits of the effective address. The Data Bank Register contains the high-order 8 bits of the operand address.

Instruction:	opcode	addrl	addrh
Operand Address:	DBR	addrh	addrl

3. Absolute Long—al

The second, third, and fourth byte of the instruction form the 24-bit effective address.

Instruction:	opcode	addrl	addrh	baddr
Operand Address:	baddr	addrh	addrl	

4. Direct—d

The second byte of the instruction is added to the Direct Register (D) to form the effective address. An additional cycle is required when the Direct Register is not page aligned (DL not equal 0). The Bank register is always 0

Instruction:	opcode	offset

		Direct Register
	+	offset
Operand Address:	00	effective address

5. Accumulator—A

This form of addressing always uses a single byte instruction. The operand is the Accumulator

6. Implied—i

Implied addressing uses a single byte instruction. The operand is implicitly defined by the instruction.

7. Direct Indirect Indexed—(d),y

This address mode is often referred to as Indirect,Y. The second byte of the instruction is added to the Direct Register (D). The 16-bit contents of this memory location is then combined with the Data Bank register to form a 24-bit base address. The Y Index Register is added to the base address to form the effective address.

Instruction:	opcode	offset

		Direct Register
	+	offset
	00	direct address
then:		
	00	(direct address)
	+ DBR	
	base address	
	+	Y Reg
Operand Address:	effective address	

8. Direct Indirect Long Indexed—[d],y

With this address mode, the 24-bit base address is pointed to by the sum of the second byte of the instruction and the Direct Register. The effective address is this 24-bit base address plus the Y Index Register.

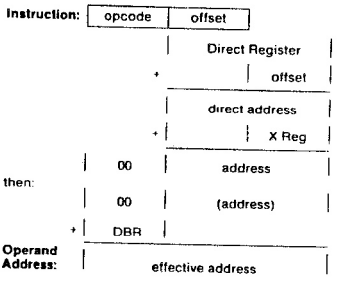
Instruction:	opcode	offset

		Direct Register
	+	offset
	00	direct address
then:		
	(direct address)	
	+	Y Reg
Operand Address:	effective address	

9. Direct Indexed Indirect—(d,x)

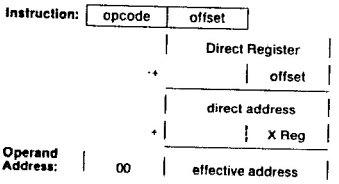
This address mode is often referred to as Indirect,X. The second byte of the instruction is added to the sum of the Direct Register and the X Index Register. The result points to the low-order 16 bits of the effective address. The Data Bank Register contains the high-order 8 bits of the effective address.

APPENDIX I (continued)



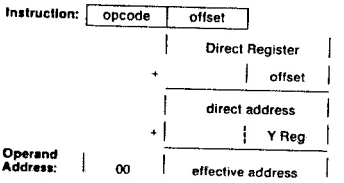
10. Direct Indexed With X—d,x

The second byte of the instruction is added to the sum of the Direct Register and the X Index Register to form the 16-bit effective address. The operand is always in Bank 0.



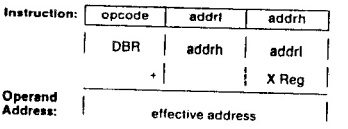
11. Direct Indexed With Y—d,y

The second byte of the instruction is added to the sum of the Direct Register and the Y Index Register to form the 16-bit effective address. The operand is always in Bank 0.



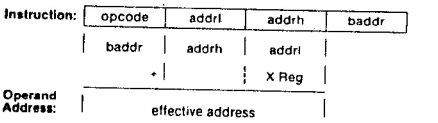
12. Absolute Indexed With X—a,x

The second and third bytes of the instruction are added to the X Index Register to form the low-order 16 bits of the effective address. The Data Bank Register contains the high-order 8 bits of the effective address.



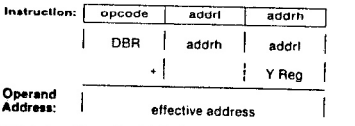
13. Absolute Long Indexed With X—al,x

The second, third and fourth bytes of the instruction form a 24-bit base address. The effective address is the sum of this 24-bit address and the X Index Register.



14. Absolute Indexed With Y—a,y

The second and third bytes of the instruction are added to the Y Index Register to form the low-order 16 bits of the effective address. The Data Bank Register contains the high-order 8 bits of the effective address.



15. Program Counter Relative—r

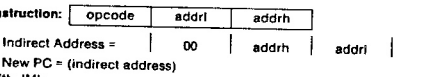
This address mode, referred to as Relative Addressing, is used only with the Branch instructions. If the condition being tested is met, the second byte of the instruction is added to the Program Counter, which has been updated to point to the opcode of the next instruction. The offset is a signed 8-bit quantity in the range from -128 to 127. The Program Bank Register is not affected.

16. Program Counter Relative Long—rl

This address mode, referred to as Relative Long Addressing, is used only with the Unconditional Branch Long instruction (BRL) and the Push Effective Relative instruction (PER). The second and third bytes of the instruction are added to the Program Counter, which has been updated to point to the opcode of the next instruction. With the branch instruction, the Program Counter is loaded with the result. With the Push Effective Relative instruction, the result is stored on the stack. The offset is a signed 16-bit quantity in the range from -32768 to 32767. The Program Bank Register is not affected.

17. Absolute Indirect—(a)

The second and third bytes of the instruction form an address to a pointer in Bank 0. The Program Counter is loaded with the first and second bytes at this pointer. With the Jump Long (JML) instruction, the Program Bank Register is loaded with the third byte of the pointer.



New PC = (indirect address)

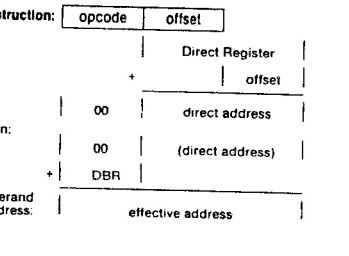
with JML:

New PC = (indirect address)

New PBR = (indirect address +2)

18. Direct Indirect—(d)

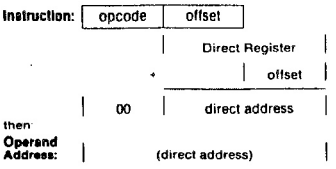
The second byte of the instruction is added to the Direct Register to form a pointer to the low-order 16 bits of the effective address. The Data Bank Register contains the high-order 8 bits of the effective address.



APPENDIX I (continued)

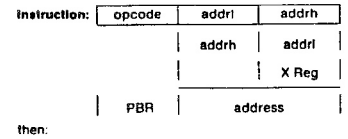
19. Direct Indirect Long—[d]

The second byte of the instruction is added to the Direct Register to form a pointer to the 24-bit effective address.



20. Absolute Indexed Indirect—(a,x)

The second and third bytes of the instruction are added to the X Index Register to form a 16-bit pointer in Bank 0. The contents of this pointer are loaded in the Program Counter. The Program Bank Register is not changed.

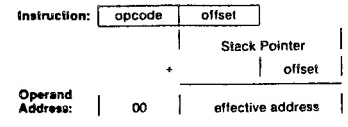


21. Stack—s

Stack addressing refers to all instructions that push or pull data from the stack, such as Push, Pull, Jump to Subroutine, Return from Subroutine, Interrupts, and Return from Interrupt. The bank address is always 0. Interrupt Vectors are always fetched from Bank 0.

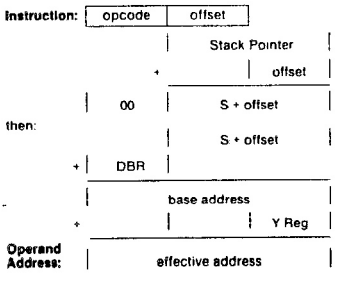
22. Stack Relative—d,s

The low-order 16 bits of the effective address is formed from the sum of the second byte of the instruction and the Stack Pointer. The high-order 8 bits of the effective address is always zero. The relative offset is an unsigned 8-bit quantity in the range of 0 to 255.



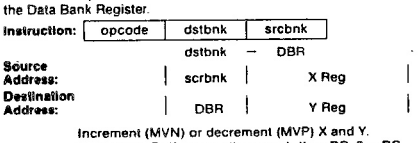
23. Stack Relative Indirect Indexed—(d,s),y

The second byte of the instruction is added to the Stack Pointer to form a pointer to the low-order 16-bit base address in Bank 0. The Data Bank Register contains the high-order 8 bits of the base address. The effective address is the sum of the 24-bit base address and the Y Index Register.



24. Block Source Bank, Destination Bank—xyc

This addressing mode is used by the Block Move instructions. The second byte of the instruction contains the high-order 8 bits of the destination address. The Y index Register contains the low-order 16 bits of the destination address. The third byte of the instruction contains the high-order 8 bits of the source address. The X Index Register contains the low-order 16 bits of the source address. The C Accumulator contains one less than the number of bytes to move. The second byte of the block move instructions is also loaded into the Data Bank Register.



Increment (MVN) or decrement (MVP) X and Y.
Decrement C (if greater than zero), then PC+3 → PC.

APPENDIX J

Addressing Mode Summary

Address Mode	Instruction Times In Memory Cycles		Memory Utilization In Number of Program Sequence Bytes	
	Original 8 Bit NMOS 6502	New W65C816	Original 8 Bit NMOS 6502	New W65C816
1. Immediate	2	2[3]	2	2[3]
2. Absolute	4[5]	4[3,5]	3	3
3. Absolute Long	—	5[3]	—	4
4. Direct	3[5]	3[3,4,5]	2	2
5. Accumulator	2	2	1	1
6. Implied	2	2	1	1
7. Direct Indirect Indexed (IND), Y	5[1]	5[1,3,4]	2	2
8. Direct Indirect Indexed Long (IND), Y Long	—	6[3,4]	—	2
9. Direct Indexed Indirect (IND, X)	6	6[3,4]	2	2
10. Direct, X	4[5]	4[3,4,5]	2	2
11. Direct, Y	4	4[3,4]	2	2
12. Absolute, X	4[1,5]	4[1,3,5]	3	3
13. Absolute Long, X	—	5[3]	—	4
14. Absolute, Y	4[1]	4[1,3]	3	3
15. Relative	2[1,2]	2[2]	2	2
16. Relative Long	—	3[2]	—	3
17. Absolute Indirect (Jump)	5	5	3	3
18. Direct Indirect	—	5[3,4]	—	2
19. Direct Indirect Long	—	6[3,4]	—	2
20. Absolute Indexed Indirect (Jump)	—	6	—	3
21. Stack	3-7	3-8	1-3	1-4
22. Stack Relative	—	4[3]	—	2
23. Stack Relative Indirect Indexed	—	7[3]	—	2
24. Block Move X, Y, C (Source, Destination, Block Length)	—	7	—	3

NOTES:
1. Page boundary, add 1 cycle if page boundary is crossed when forming address.
2. Branch taken, add 1 cycle if branch is taken.
3. M = 0 or X = 0, 16 bit operation, add 1 cycle, add 1 byte for immediate.
4. Direct register low (DL) not equal zero, add 1 cycle.
5. Read-Modify-Write, add 2 cycles for M = 1, add 3 cycles for M = 0.

NOTES

NOTES