APPLE II
Reference Manual
January 1978 - 2008

Good artists copy. Great artists steal.

Pablo Picasso
# TABLE OF CONTENTS

## A. GETTING STARTED WITH YOUR APPLE II

1. Unpacking ........................................ 1  
2. Warranty Registration Card ..................... 1  
3. Check for Shipping Damage ..................... 2  
4. Power Up ......................................... 2  
5. APPLE II Speaks Several Languages .......... 2  
6. APPLE Integer BASIC ............................ 3  
7. Running Your First and Second Programs ...... 3  
8. Running 16K Startrek ............................ 3  
9. Loading a Program Tape ........................ 4  
10. Breakout and Color Demos Tapes .............. 6  
11. Breakout and Color Demos Program Listings .. 12  
12. How to Play Startrek ........................... 14  
13. Loading HIRES Demo Tape ...................... 15  

## B. APPLE II INTEGER BASIC

1. BASIC Commands .................................. 18  
2. BASIC Operators .................................. 19  
3. BASIC Functions .................................. 22  
4. BASIC Statements .................................. 23  
5. Special Control and Editing .................... 28  
6. Table A - Graphics Colors ...................... 29  
7. Special Controls and Features .................. 30  
8. BASIC Error Messages ............................ 32  
9. Simplified Memory Map ........................... 33  
10. Data Read/Save Subroutines .................... 34  
11. Simple Tone Subroutines ........................ 43  
12. High Resolution Graphics Subroutines and Listings 46

## C. APPLE II Firmware

1. System Monitor Commands ....................... 68  
2. Control and Editing Characters ............... 72  
3. Special Controls and Features ................ 74  
4. Annotated Monitor and Disassembler Listing .. 76  
5. Binary Floating Point Package .................. 94  
6. Sweet 16 Interpreter Listing ................... 96  
7. 6502 Op Codes ..................................... 100  

## D. APPLE II HARDWARE

1. Getting Started with Your APPLE II Board ..... 107  
2. APPLE II Switching Power Supply .............. 110  
3. Interfacing with the Home TV ................... 112  
4. Simple Serial Output ............................. 114  
5. Interfacing the APPLE - Signals, Loading, Pin Connections 122  
6. Memory - Options, Expansion, Map, Address .. 133  
7. System Timing ..................................... 140  
8. Schematics ....................................... 141

## 13. Additional BASIC Program Examples

a. Rod's Color Pattern (4K) ................. 55  
   b. Pong (4K) .................................. 56  
   c. Color Sketch (4K) ......................... 57  
   d. Mastermind (8K) .......................... 59  
   e. Biorhythm (4K) ......................... 61  
   f. Dragon Maze (4K) ...................... 63

---

[1] Getting Started with Your APPLE II Board  
[2] APPLE II Switching Power Supply  
[3] Interfacing with the Home TV  
[4] Simple Serial Output  
[5] Interfacing the APPLE - Signals, Loading, Pin Connections  
[8] Schematics
GETTING STARTED WITH YOUR APPLE II

Unpacking

Don't throw away the packing material. Save it for the unlikely event that you may need to return your Apple II for warrantee repair. If you bought an Apple II Board only, see hardware section in this manual on how to get started. You should have received the following:

1. Apple II system including mother printed circuit board with specified amount of RAM memory and 8K of ROM memory, switching power supply, keyboard, and case assembly.

2. Accessories Box including the following:
   a. This manual including warranty card.
   b. Pair of Game Paddles
   c. A.C. Power Cord
   d. Cassette tape with "Breakout" on one side and "Color Demos" on the other side.
   e. Cassette recorder interface cable (miniature phone jack type)

3. If you purchased a 16K or larger system, your accessory box should also contain:
   a. 16K Startrek game cassette with High Resolution Graphics Demo ("HIRES") on the flipside.
   b. Applesoft Floating Point Basic Language Cassette with an example program on the other side.
   c. Applesoft reference manual

4. In addition other items such as a vinyl carrying case or hobby board peripheral may have been included if specifically ordered as "extras".

Notify your dealer or Apple Computer, Inc. immediately if you are missing any items.

Warranty Registration Card

Fill this card out immediately and completely and mail to Apple in order to register for one year warranty and to be placed on owners club mailing list. Your Apple II's serial number is located on the bottom near the rear edge. You model number is:

A2SØØMX

MM is the amount of memory you purchased. For Example:

A2SØØ08X

is an 8K Byte Apple II system.
Check for Damage

Inspect the outside case of your Apple for shipping damage. Gently lift up on the top rear of the lid of the case to release the lid snaps and remove the lid. Inspect the inside. Nothing should be loose and rattling around. Gently press down on each integrated circuit to make sure that each is still firmly seated in its socket. Plug in your game paddles into the Apple II board at the socket marked "GAME I/O" at location J14. See hardware section of this manual for additional detail. The white dot on the connector should be face forward. Be careful as this connector is fragile. Replace the lid and press on the back top of it to re-snap it into place.

Power Up

First, make sure that the power ON/OFF switch on the rear power supply panel on your Apple II is in the "OFF" position. Connect the A.C. power cord to the Apple and to a 3 wire 12Ø volt A.C. outlet. Make sure that you connect the third wire to ground if you have only a two conductor house wiring system. This ground is for your safety if there is an internal failure in the Apple power supply, minimizes the chance of static damage to the Apple, and minimizes RFI problems.

Connect a cable from the video output jack on the back of the Apple to a TV set with a direct video input jack. This type of set is commonly called a "Monitor". If your set does not have a direct video input, it is possible to modify your existing set. Write for Apple's Application note on this. Optionally you may connect the Apple to the antenna terminals of your TV if you use a modulator. See additional details in the hardware section of this manual under "Interfacing with the Home TV".

Now turn on the power switch on the back of the Apple. The indicator light (it's not a switch) on the keyboard should now be ON. If not, check A.C. connections. Press and release the "Reset" button on the keyboard. The following should happen: the Apple's internal speaker should beep, an asterisk ("*"") prompt character should appear at the lower left hand corner of your TV, and a flashing white square should appear just to the right of the asterisk. The rest of the TV screen will be made up of random text characters (typically question marks).

If the Apple beeps and garbage appears but you cannot see an "*" and the cursor, the horizontal or vertical height settings on the TV need to be adjusted. Now depress and release the "ESC" key, then hold down the "SHIFT" key while depressing and releasing the P key. This should clear your TV screen to all black. Now depress and release the "RESET" key again. The "*" prompt character and the cursor should return to the lower left of your TV screen.
Apple Speaks Several Languages

The prompt character indicates which language your Apple is currently in. The current prompt character, an asterisk ("*"), indicates that you are in the "Monitor" language, a powerful machine level language for advanced programmers. Details of this language are in the "Firmware" section of this manual.

Apple Integer BASIC

Apple also contains a high level English oriented language called Integer BASIC, permanently in its ROM memory. To switch to this language hold down the "CTRL" key while depressing and releasing the "B" key. This is called a control-B function and is similar to the use of the shift key in that it indicates a different function to the Apple. Control key functions are not displayed on your TV screen but the Apple still gets the message. Now depress and release the "RETURN" key to tell Apple that you have finished typing a line on the keyboard. A right facing arrow (">") called a caret will now appear as the prompt character to indicate that Apple is now in its Integer BASIC language mode.

Running Your First and Second Program

Read through the next three sections that include:

1. Loading a BASIC program Tape
2. Breakout Game Tape
3. Color Demo Tape

Then load and run each program tape. Additional information on Apple II's Integer BASIC is in the next section of this manual.

Running 16K Startrek

If you have 16K Bytes or larger memory in your Apple, you will also receive a "STARTREK" game tape. Load this program just as you did the previous two, but before you "RUN" it, type in "HIMEM: 16384" to set exactly where in memory this program is to run.
LOADING A PROGRAM TAPE

INTRODUCTION

This section describes a procedure for loading BASIC programs successfully into the Apple II. The process of loading a program is divided into three sections: System Checkout, Loading a Tape and What to do when you have Loading Problems. They are discussed below.

When loading a tape, the Apple II needs a signal of about 2 1/2 to 5 volts peak-to-peak. Commonly, this signal is obtained from the "Monitor" or "earphone" output jack on the tape recorder. Inside most tape recorders, this signal is derived from the tape recorder's speaker. One can take advantage of this fact when setting the volume levels. Using an Apple Computer pre-recorded tape, and with all cables disconnected, play the tape and adjust the volume to a loud but un-distorted level. You will find that this volume setting will be quite close to the optimum setting.

Some tape recorders (mostly those intended for use with hi-fi sets) do not have an "earphone" or high-level "monitor" output. These machines have outputs labeled "line output" for connection to the power amplifier. The signal levels at these outputs are too low for the Apple II in most cases.

Cassette tape recorders in the $40 - $50 range generally have ALC (Automatic Level Control) for recording from the microphone input. This feature is useful since the user doesn't have to set any volume controls to obtain a good recording. If you are using a recorder which must be adjusted, it will have a level meter or a little light to warn of excessive recording levels. Set the recording level to just below the level meter's maximum, or to just a dim indication on the level lamp. Listen to the recorded tape after you've saved a program to ensure that the recording is "loud and clear".

Apple Computer has found that an occasional tape recorder will not function properly when both Input and Output cables are plugged in at the same time. This problem has been traced to a ground loop in the tape recorder itself which prevents making a good recording when saving a program. The easiest solution is to unplug the "monitor" output when recording. This ground loop does not influence the system when loading a pre-recorded tape.
Tape recorder head alignment is the most common source of tape recorder problems. If the playback head is skewed, then high frequency information on pre-recorded tapes is lost and all sorts of errors will result. To confirm that head alignment is the problem, write a short program in BASIC. >10 END is sufficient. Then save this program. And then rewind and load the program. If you can accomplish this easily but cannot load pre-recorded tapes, then head alignment problems are indicated.

Apple Computer pre-recorded tapes are made on the highest quality professional duplicating machines, and these tapes may be used by the service technician to align the tape recorder's heads. The frequency response of the tape recorder should be fairly good; the 6 KHz tone should be not more than 3 db down from a 1 KHz tone, and a 9 KHz tone should be no more than 9 db down. Note that recordings you have made yourself with mis-aligned heads may not play properly with the heads properly aligned. If you made a recording with a skewed record head, then the tiny magnetic fields on the tape will be skewed as well, thus playing back properly only when the skew on the tape exactly matches the skew of the tape recorder's heads. If you have saved valuable programs with a skewed tape recorder, then borrow another tape recorder, load the programs with the old tape recorder into the Apple, then save them on the borrowed machine. Then have your tape recorder properly aligned.

Listening to the tape can help solve other problems as well. Flaws in the tape, excessive speed variations, and distortion can be detected this way. Saving a program several times in a row is good insurance against tape flaws. One thing to listen for is a good clean tone lasting for at least 3 1/2 seconds is needed by the computer to "set up" for proper loading. The Apple puts out this tone for about 10 seconds when saving a program, so you normally have 6 1/2 seconds of leeway. If the playback volume is too high, you may pick up tape noise before getting to the set-up tone. Try a lower playback volume.

SYSTEM CHECKOUT

A quick check of the Apple II computer system will help you spot any problems that might be due to improperly placed or missing connections between the Apple II, the cassette interface, the Video display, and the game paddles. This checkout procedure takes just a few seconds to perform and is a good way of insuring that everything is properly connected before the power is turned on.
1. **POWER TO APPLE** - check that the AC power cord is plugged into an appropriate wall socket, which includes a "true" ground and is connected to the Apple II.

2. **CASSETTE INTERFACE** - check that at least one cassette cable double ended with miniature phone tip jacks is connected between the Apple II cassette Input port and the tape recorder's MONITOR plug socket.

3. **VIDEO DISPLAY INTERFACE** -
   a) for a video monitor - check that a cable connects the monitor to the Apple's video output port.
   b) for a standard television - check that an adapter (RF modulator) is plugged into the Apple II (either in the video output (K14) or the video auxiliary socket (J148), and that a cable runs between the television and the Adapter's output socket.

4. **GAME PADDLE INTERFACE** - if paddles are to be used, check that they are connected into the Game I/O connector (J14) on the right-hand side of the Apple II mainboard.

5. **POWER ON** - flip on the power switch in back of the Apple II, the "power" indicator on the keyboard will light. Also make sure the video monitor (or TV set) is turned on.

After the Apple II system has been powered up and the video display presents a random matrix of question marks or other text characters the following procedure can be followed to load a BASIC program tape:

1. Hit the **RESET** key. An asterick, "*", should appear on the lefthand side of the screen below the random text pattern. A flashing white cursor will appear to the right of the asterick.

2. Hold down the **CTRL** key, depress and release the **B** key, then depress the "**RETURN**" key and release the "**CTRL**" key. A right facing arrow should appear on the lefthand side of the screen with a flashing cursor next to it. If it doesn't, repeat steps 1 and 2.

3. Type in the word "**LOAD**" on the keyboard. You should see the word in between the right facing arrow and the flashing cursor. Do not depress the "**RETURN**" key yet.

4. Insert the program cassette into the tape recorder and rewind it.

5. If not already set, adjust the Volume control to 50-70% maximum. If present, adjust the Tone control to 80-100% maximum.
6. Start the tape recorder in "PLAY" mode and now depress the "RETURN" key on the Apple II.

7. The cursor will disappear and Apple II will beep in a few seconds when it finds the beginning of the program. If an error message is flashed on the screen, proceed through the steps listed in the Tape Problem section of this paper.

8. A second beep will sound and the flashing cursor will reappear after the program has been successfully loaded into the computer.

9. Stop the tape recorder. You may want to rewind the program tape at this time.

10. Type in the word "RUN" and depress the "RETURN" key.

The steps in loading a program have been completed and if everything has gone satisfactorily the program will be operating now.

LOADING PROBLEMS

Occasionally, while attempting to load a BASIC program Apple II beeps and a memory full error is written on the screen. At this time you might wonder what is wrong with the computer, with the program tape, or with the cassette recorder. Stop. This is the time when you need to take a moment and checkout the system rather than haphazardly attempting to resolve the loading problem. Thoughtful action taken here will speed in a program's entry. If you were able to successfully turn on the computer, reset it, and place it into BASIC then the Apple II is probably operating correctly. Before describing a procedure for resolving this loading problem, a discussion of what a memory full error is in order.

The memory full error displayed upon loading a program indicates that not enough (RAM) memory workspace is available to contain the incoming data. How does the computer know this? Information contained in the beginning of the program tape declares the record length of the program. The computer reads this data first and checks it with the amount of free memory. If adequate workspace is available program loading continues. If not, the computer beeps to indicate a problem, displays a memory full error statement, stops the loading procedure, and returns command of the system to the keyboard. Several reasons emerge as the cause of this problem.
Memory Size too Small

Attempting to load a 16K program into a 4K Apple II will generate this kind of error message. It is called loading too large of a program. The solution is straight forward: only load appropriately sized programs into suitably sized systems.

Another possible reason for an error message is that the memory pointers which indicate the bounds of available memory have been preset to a smaller capacity. This could have happened through previous usage of the "HIMEN:" and "LOMEN:" statements. The solution is to reset the pointers by BC (CTRL B) command. Hold the CTRL key down, depress and release the B key, then depress the RETURN key and release the CTRL key. This will reset the system to maximum capacity.

Cassette Recorder Inadjustment

If the Volume and Tone controls on the cassette recorder are not properly set a memory full error can occur. The solution is to adjust the Volume to 50-70% maximum and the Tone (if it exists) to 80-100% maximum.*

A second common recorder problem is skewed head azimuth. When the tape head is not exactly perpendicular to the edges of the magnetic tape some of the high frequency data on tape can be skipped. This causes missing bits in the data sent to the computer. Since the first data read is record length an error here could cause a memory full error to be generated because the length of the record is inaccurate. The solution: adjust tape head azimuth. It is recommended that a competent technician at a local stereo shop perform this operation. Often times new cassette recorders will not need this adjustment.

*Apple Computer Inc. has tested many types of cassette recorders and so far the Panasonic RQ-309 DS (less than $40.00) has an excellent track record for program loading.
Tape Problems

A memory full error can result from unintentional noise existing in a program tape. This can be the result of a program tape starting on its header which sometimes causes a glitch going from a nonmagnetic to magnetic recording surface and is interpreted by the computer as the record length. Or, the program tape can be defective due to false erasure, imperfections in the tape, or physical damage. The solution is to take a moment and listen to the tape. If any imperfections are heard then replacement of the tape is called for. Listening to the tape assures that you know what a "good" program tape sounds like. If you have any questions about this please contact your local dealer or Apple for assistance.

If noise or a glitch is heard at the beginning of a tape advance the tape to the start of the program and re-Load the tape.

Dealing with the Loading Problem

With the understanding of what a memory full error is an efficient way of dealing with program tape loading problems is to perform the following procedure:

1. Check the program tape for its memory requirements. Be sure that you have a large enough system.

2. Before loading a program reset the memory pointers with the Bc (control B) command.

3. In special cases have the tape head azimuth checked and adjusted.

4. Check the program tape by listening to it. a) Replace it if it is defective, or b) start it at the beginning of the program.

5. Then re-LOAD the program tape into the Apple II.

In most cases if the preceding is followed a good tape load will result.

UNSOLVED PROBLEMS

If you are having any unsolved loading problems, contact your nearest local dealer or Apple Computer Inc.
PROGRAM DESCRIPTION
Breakout is a color graphics game for the Apple II computer. The object of the game is to "knock-out" all 160 colored bricks from the playing field by hitting them with the bouncing ball. You direct the ball by hitting it with a paddle on the left side of the screen. You control the paddle with one of the Apple's Game Paddle controllers. But watch out: you can only miss the ball five times:

There are eight columns of bricks. As you penetrate through the wall the point value of the bricks increases. A perfect game is 720 points; after five balls have been played the computer will display your score and a rating such as "Very Good", "Terrible!", etc. After ten hits of the ball, its speed with double, making the game more difficult. If you break through to the back wall, the ball will rebound back and forth, racking up points.

Breakout is a challenging game that tests your concentration, dexterity, and skill.

REQUIREMENTS
This program will fit into a 4K or greater system.
BASIC is the programming language used.

PLAYING BREAKOUT

1. Load Breakout game following instructions in the "Loading a BASIC Program from Tape" section of this manual.
2. Enter your name and depress RETURN key.
3. If you want standard BREAKOUT colors type in Y or Yes and hit RETURN. The game will then begin.
4. If the answer to the previous questions was N or No then the available colors will be displayed. The player will be asked to choose colors, represented by a number from 0 to 15, for background, even bricks, odd bricks, paddle and ball colors. After these have been chosen the game will begin.
5. At the end of the game you will be asked if they want to play again. A Y or Yes response will start another game. A N or No will exit from the program.

NOTE: A game paddle (150k ohm potentiometer) must be connected to PDL (Ø) of the Game I/O connector for this game.

COLOR DEMO TAPE

PROGRAM DESCRIPTION

COLOR DEMO demonstrates some of the Apple II video graphics capabilities. In it are ten examples: Lines, Cross, Weaving, Tunnel, Circle, Spiral, Tones, Spring, Hyperbola, and Color Bars. These examples produce various combinations of visual patterns in fifteen colors on a monitor or television screen. For example, Spiral combines colorgraphics with tones to produce some amusing patterns. Tones illustrates various sounds that you can produce with the two inch Apple speaker. These examples also demonstrate how the paddle inputs (PDL(X)) can be used to control the audio and visual displays. Ideas from this program can be incorporated into other programs with a little modification.

REQUIREMENTS

4K or greater Apple II system, color monitor or television, and paddles are needed to use this program. BASIC is the programming language used.
BREAKOUT GAME
PROGRAM LISTING

PROGAM LISTING

5 GOTO 15
10 B=C: GOSUB (B)=-20/B: IF 0<0 THEN 5: PRINT 34,21; S: PRINT "VLINE 0,0+5 AT 0: COLOR=R:
   IF X=Y THEN 175: IF 0 THEN VLINE 0,0-1 AT 0;P=R: RETURN
15 DIM A(15),B(10),X,R=16:10: C=0:G=7:15: TEXT 2 CALL -
   936: VTABLE 4: TAB 10: PRINT "**** BREAKOUT ****": PRINT
   20 PRINT "OBJECT IS TO DESTROY ALL BRICKS": PRINT : INPUT
   "HI, WHAT'S YOUR NAME?",A,R
25 PRINT "STANDARD COLORS": PRINT
   1: INPUT "Y/N?",B,R: G: CALL -
   936: IF B(1,1)"N" THEN 40 ELSE 10 = FOR I=0 TO 39: COLOR=I/2*
   (1<32) VLINE 0,0 AT I
30 NEXT I: POKE 34,20: PRINT :
   PRINT "PROGRAM LISTING": PRINT I=0 TO 10: VTABLE 21+1 MOD 2+ TAB 1+ I=I+1 PRINT I: I=I+1 NEXT I: POKE
   34,22: VTABLE 24: PRINT "BACKGROUND":
35 GOSUB 95:E=1: PRINT "EVEN BRICK":
   :GOSUB 95:E=1: PRINT "GOOD BRICK" E:
   :GOSUB 95:E=1: PRINT "BAD BRICK" E:
   :GOSUB 95:E=1: PRINT "BALL":
   :GOSUB 95
40 POKE 34,239: COLOR=0: FOR I= 0 TO 39: VLINE 0,0 AT I: NEXT
   1: FOR I=29 TO 34 STEP 2: TAB I=I+1 PRINT I/2-39; COLOR=R:
   VLINE 0,39 AT I: COLOR=R: FOR J=1 MOD 4 TO 39 STEP 4
   45 bin J,j+1 AT I: NEXT J,i: TAB 5: PRINT "SCORE =" S: PRINT
   1: PRINT : POKE 34,21;5:S=IF
   5=S=5:X=19:Y=19:Z=6
   50 COLOR=R: PLOT X,Y,X-X,Y= RND (120):V=1:U= RND 5-1:V=1 L-1: I= L+100: TAB
   60 IF L=1 THEN L=0: TAB 61: IF L=1 THEN PRINT L;" BALLS LEFT":
   55 IF L=1 THEN PRINT "LAST BALL","": PRINT : FOR I=1 TO 100
   65 SUB 10: NEXT 1: N=0
60 J,Y,W: IF J=0 AND J(120) THEN
   65 SUB:W=J:Y: FOR I=1 TO 6:X= PECK (-16336): NEXT 1
   65 I=X+Y: IF I<0 THEN 100: GOSUB
   170: COLOR=I*X=J/3: IF 129
   THEN 93: SIMPLE(1)=X=A THEN
   85: IF I THEN 100:N=1:N=1 V=129
   5):W=1:K=P-2:5=N=1
   70 Z= PECK (-16336) PECK (-16336)
   + PECK (-16336) PECK (-16336)
   + PECK (-16336): GOTO 35
   75 FOR I=1 TO 6:X= PECK (-16336)
   80 W=V
   85 POKE X,Y,Z: COLOR=Z: PLOT 1, X:Y:Z: COLOR=1: GOTO 60
   90 PRINT "INVALID! REENTER":
   95 INPUT "COLOR (0 TO 15):",E:
   100 IF E=0 OR E=15 THEN 90: RETURN
   100 IF N THEN V= ABS (Y): VLINE
   X-2,Y-2+1 AT 11 AND 11-2-
   9: VTABLE 21: TAB 1: PRINT 5
   105 S= PECK (-16336) PECK (-16336)
   + PECK (-16336): PECK (-16336)
   + PECK (-16336): PECK (-16336)
   + PECK (-16336): PECK (-16336)
   + PECK (-16336): PECK (-16336)
   110 IF S<720 THEN 180
   115 PRINT "CONGRATULATIONS: "",AS:
   120 PRINT "YOU WIN": GOTO 160
   125 PRINT "YOUR SCORE IS "",S": IS "",AS
   130 GOTO 125:S=160
   135 PRINT "TERRIBLE!": GOTO 160
   136 PRINT "LOWLY": GOTO 160
   135 PRINT "POOR": GOTO 160
   140 PRINT "FAIR": GOTO 160
   145 PRINT "GOOD": GOTO 160
   150 PRINT "VERY GOOD": GOTO 160
   155 PRINT "EXCELLENT": GOTO 160
   160 PRINT "NEARLY PERFECT.
   165 PRINT "ANOTHER GAME "",A,": (Y/N)
   170 OR (A): GOSUB (B)=-20/B: IF 0<0 THEN
   0=0: IF 0=0 THEN 0=0: COLOR=0: VLINE 0,0+5 AT 0: COLOR=0:
   IF P=0 THEN 175: IF P THEN VLINE 0,0-1 AT 0;P=0: RETURN
   175 IF P=0 THEN RETURN : IF 0=0 THEN VLINE 0,0+5 AT 0;P=0:
   RETURN
   180 FOR I=1 TO 0:SUB: PECK (-16336)
   NEXT I: GOTO 50
COLOR DEMO PROGRAM LISTING

10 DIM C(4); POKE 2,173: POKE
3,48: POKE 4,192: POKE 5,165
: POKE 6,8: POKE 7,32: POKE
8,168: POKE 9,252: POKE 10,
165: POKE 11,1: POKE 12,268

20 POKE 13,4: POKE 14,198: POKE
15,24: POKE 16,240: POKE 17
,5: POKE 18,198: POKE 19,1:
POKE 20,78: POKE 21,2: POKE
22,0: POKE 23,36

30 TEXT = CALL -936: VTAB 4: TAB
8: PRINT "4K COLOR DEMOS": PRINT
: PRINT "1 LINES": PRINT "2 CROS
5": PRINT "3 WEAVING"

40 PRINT "4 TUNNEL": PRINT "5 CIRO
E": PRINT "6 SPIRAL **": PRINT
"7 TONES **": PRINT "8 SPRING"

50 PRINT "9 HYPERBOLA": PRINT
"10 COLOR BARS": PRINT " PRINT
** NEEDS PDL(0) CONNECTED" :
PRINT

60 PRINT "HIT ANY KEY FOR NEW DEMO"
: z=0: PRINT " INPUT "WHICH DEMO
# " ,I: GR : IF I>0 AND I<11
THEN GOTO 100+I: GOTO 30

70 INPUT "WHICH DEMO WOULD YOU LIKE
",I: GR : IF I AND I<20 THEN
GOTO 100+I: GOTO 30

100 =I+1 MOD 79;1=I(1<139)X(79
-1): GOSUB 2000: GOSUB 10000
: GOTO 100

200 =I+1 MOD 39;J=I: GOSUB 2000
: J=39-1: GOSUB 2000: GOSUB
10000: GOTO 200

300 J=J+1:J=J MOD 22+1: FOR I=1
TO 1295: COLOR=I MOD 7: PLOT
:(2*I) MOD 37,(3+I) MOD 35: NEXT
I: GOSUB 10000: GOTO 300

400 FOR I=1 TO 4:C(I)= RND (16)
 : NEXT I

410 FOR I=0 TO I STEP -1;C(I+1)
 =C(I): NEXT I: C(I)= RND (16
 :): FOR I=1 TO 5: FOR I=1 TO
4

420 COLOR=C(I);L=;X=5+14+1=X;33-
L: HLIN K,L AT K: VLIN K,L AT
L: HLIN L,K AT L: VLIN L,K AT
K: NEXT J;I: GOSUB 10000: GOTO
410

500 Z=2:GOTO 900

600 COLOR= RND (16): FOR I=0 TO
18 STEP 2:J=39-I: HLIN I,J AT
I: GOSUB 646: VLIN I,J AT J:
GOSUB 646

610 HLIN I+2,J AT J: GOSUB 646:
VLIN I+2,J AT I+2: GOSUB 646
 : NEXT I

620 COLOR= RND (16): FOR I=18 TO
0 STEP -2;J=39-I: HLIN I+2,
J AT I+2: GOSUB 646: HLIN I+
2,J AT J: GOSUB 646

630 VLIN I,J AT J: GOSUB 646: HLIN
1,J AT I: GOSUB 646: NEXT I:
GOSUB 10000: GOTO 600

640 K=I+7;L=K*K+5*K+26+70: L=32767
: L+( PDL (0) )= POK E 0,K:
: POKE 1,L MOD 256: POKE 24,
L/256+1: CALL 2: RETURN

700 I=RND (38)+1:J=I+I5+1+I26+
70;I=32767/J*K PDL (0)/10):
POKE 0,1: POKE 1,K MOD 256
: POKE 24,(K+255)+1: CALL 2
: GOSUB 10000: GOTO 780

800 X=3;A=1000;P=1;L=20;V=4;Y=8
: J=1: COLOR=6: HLIN 0,39 AT
4: COLOR=3: GOSUB 880: COLOR=12:
VLIN 5,5: AT X

810 N=2*M-P-A/W: COLOR=0: GOSUB
880: VLIN 5,39 AT X:X=X+1: IF
X=39 THEN 626:X=3: VLIN 5,39
: AT I: GOSUB 5,39 AT 2

820 P=AH+1:Y=A/B: COLOR=12: GOSUB
880: COLOR=9: VLIN 5,3-2 AT
X: COLOR=15: PLOT X-2,M: FOR
1=0 TO J; NEXT I: GOSUB 10000
: GOTO 810

890 M=L-Y;L=L-1:Z=H+1: VLIN L1,
L2 AT X-1: VLIN L1,L2 AT X:
VLIN L1,L2 AT X: RETURN

900 I=I+1 MOD 15: FOR Y=0 TO 35
 : FOR X=0 TO 39: COLOR=I+1 ABS
(29-X-2)*: ABS (26-Y-2)/25
 : PLOT X,Y: NEXT X,Y: GOSUB
10000: GOTO 900

1000 CALL -936

1010 J=1+J MOD 32: COLOR=J/2: VLIN
0,39 AT 3+J: VTAB 21+(J-2) MOD
2: TAB 3+J: IF J MOD 2 THEN
PRINT J: GOSUB 10000: GOTO
1010

2000 COLOR= RND (16): HLIN 0,39 AT
J: COLOR= RND (16): VLIN 0,
39 AT J: RETURN

10000 IF PEEK (-16384)/128 THEN RETURN
: POKE -16388,8: POP : GOTO
30
THIS IS A SHORT DESCRIPTION OF HOW TO PLAY STAR TREK ON THE APPLE COMPUTER.

THE UNIVERSE IS MADE UP OF 64 QUADRANTS IN AN 8 BY 8 MATRIX. THE QUADRANT IN WHICH YOU THE ENTERPRISE ARE, IS IN WHITE, AND A BLOW UP OF THAT QUADRANT IS FOUND IN THE LOWER LEFT CORNER. YOUR SPACE SHIP STATUS IS FOUND IN A TABLE TO THE RIGHT SIDE OF THE QUADRANT BLOW UP.

THIS IS A SEARCH AND DESTROY MISSION. THE OBJECT IS TO LONG-RANGE SENSE FOR INFORMATION AS TO WHERE KLINGONS (K) ARE, MOVE TO THAT QUADRANT, AND DESTROY.

NUMBERS DISPLAYED FOR EACH QUADRANT DENOTE:
* OF STARS IN THE ONES PLACE
* OF BASES IN THE TENS PLACE
* OF KLINGONS IN THE HUNDREDS PLACE

AT ANY TIME DURING THE GAME, FOR INSTANCE BEFORE ONE TOTALLY RUNS OUT OF ENERGY, OR NEEDS TO REGENERATE ALL SYSTEMS, ONE MOVES TO A QUADRANT WHICH INCLUDES A BASE, IONS NEXT TO THAT BASE (B) AT WHICH TIME THE BASE SELF-DESTRUCTS AND THE ENTERPRISE (E) HAS ALL SYSTEMS 'GO' AGAIN.

TO PLAY:
1. THE COMMANDS CAN BE OBTAINED BY TYPING A '0' (ZERO) AND RETURN. THEY ARE:
   1. PROPULSION 2. REGENERATE
   3. LONG RANGE SENSORS 4. PHASERS
   5. PHOTON TORPEDOES 6. GALAXY RECORD
   7. COMPUTER 8. PROBE
   9. SHIELD ENERGY 10. DAMAGE REPORT
   11. LOAD PHOTON TORPEDOES
2. THE COMMANDS ARE INVOKED BY TYPING THE NUMBER REFERING TO THEM FOLLOWED BY A 'RETURN'.
   A. IF RESPONSE IS 1 THE COMPUTER WILL ASK WARP OR ION AND EXPECTS 'W' IF ONE WANTS TO TRAVEL IN THE GALAXY BETWEEN QUADRANTS AND AN 'I' IF ONE WANTS ONLY INTERNAL QUADRANT TRAVEL. DURATION OF WARP FACTOR IS THE NUMBER OF SPACES OR QUADRANTS THE ENTERPRISE WILL MOVE. COURSE IS COMPASS READING IN DEGREES FOR THE DESIRED DESTINATION.
   B. A 2 REGENERATES THE ENERGY AT THE EXPENSE OF TIME.
   C. A 3 GIVES THE CONTENTS OF THE IMMEDIATE. ADJACENT QUADRANTS. THE GALAXY IS WRAP-AROUND IN ALL DIRECTIONS.
   D. A 4 FIRES PHASERS AT THE EXPENSE OF AVAILABLE ENERGY.
   E. A 5 INITIATES A SET OF QUESTIONS FOR TORPEDO FIRING. THEY CAN BE FIRED AUTOMATICALLY IF THEY HAVE BEEN LOCKED ON TARGET WHILE IN THE COMPUTER MODE, OR MAY BE FIRED MANUALLY IF THE TRAJECTORY ANGLE IS KNOWN.
   F. A 6, 8 AND 10 ALL GIVE INFORMATION ABOUT THE STATUS OF THE SHIP AND ITS ENVIRONMENT.
   G. A 9 SETS THE SHIELD ENERGY/AVAILABLE ENERGY RATIO.
   H. A 11 ASKS FOR INFORMATION ON LOADING AND UNLOADING OF PHOTON TORPEDOES AT THE EXPENSE OF AVAILABLE ENERGY.
       THE ANSWER SHOULD BE A SIGNED NUMBER. FOR EXAMPLE +5 OR -2.
   I. A 7 ENTERS A COMPUTER WHICH WILL RESPOND TO THE FOLLOWING INSTRUCTIONS:
      1. COMPUTE COURSE 2. LOCK PHASERS
      3. LOCK PHOTON TORPEDOES 4. LOCK COURSE
      5. COMPUTE TRAJECTORY 6. STATUS 7. RETURN TO COMMAND MODE
      IN THE FIRST FIVE ONE WILL HAVE TO GIVE COORDINATES.
      COORDINATES ARE GIVEN IN MATHEMATICAL NOTATION WITH THE EXCEPTION THAT THE 'Y' VALUE IS GIVEN FIRST.
      AN EXAMPLE WOULD BE 'Y,X'

COURSE OR TRAJECTORY:

0

270------------------------90

180

THIS EXPLANATION WAS WRITTEN BY ELWOOD NOT RESPONSIBLE FOR ERRORS
LOADING THE HI-RES DEMO TAPE

PROCEDURE

1. Power up system - turn the AC power switch in the back of the Apple II on. You should see a random matrix of question marks and other text characters. If you don't, consult the operator's manual for system checkout procedures.

2. Hit the RESET key. On the left hand side of the screen you should see an asterisk and a flashing cursor next to it below the text matrix.

3. Insert the HI-RES demo tape into the cassette and rewind it. Check Volume (50-70%) and Tone (80-100%) settings.

4. Type in "C00.FFFR" on the Apple II keyboard. This is the address range of the high resolution machine language sub-program. It extends from $C00 to $FFF. The R tells the computer to read in the data. Do not depress the "RETURN" key yet.

5. Start the tape recorder in playback mode and depress the "RETURN" key. The flashing cursor disappears.

6. A beep will sound after the program has been read in. STOP the tape recorder. Do not rewind the program tape yet.

7. Hold down the "CTRL" key, depress and release the B key, then depress the "RETURN" key and release the "CTRL" key. You should see a right facing arrow and a flashing cursor. The Bc command places the Apple into BASIC initializing the memory pointers.

8. Type in "LOAD", restart the tape recorder in playback mode and hit the "RETURN" key. The flashing cursor disappears. This begins the loading of the BASIC subprogram of the HI-RES demo tape.

9. A beep will sound to indicate the program is being loaded.
10. A second beep will sound, and the right facing arrow will reappear with the flashing cursor. STOP the tape recorder. Rewind the tape.

11. Type in "HIMEM:8192" and hit the "RETURN" key. This sets up memory for high resolution graphics.

12. Type in "RUN" and hit the "RETURN" key. The screen should clear and momentarily a HI-RES demo menu table should appear. The loading sequence is now completed.

SUMMARY OF HI-RES DEMO TAPE LOADING

1. RESET
2. Type in C00.FFFR
3. Start tape recorder, hit RETURN
4. Asterick or flashing cursor reappear
   Bc (CTRL B) into BASIC
5. Type in "LOAD", hit RETURN
6. BASIC prompt (7) and flashing cursor reappear. Type in "HIMEN:8192", hit RETURN
7. Type in "RUN", hit RETURN
8. STOP tape recorder, rewind tape.
APPLE II INTEGER BASIC

1. BASIC Commands
2. BASIC Operators
3. BASIC Functions
4. BASIC Statements
5. Special Control and Editing
6. Table A — Graphics Colors
7. Special Controls and Features
8. BASIC Error Messages
9. Simplified Memory Map
10. Data Read Save Subroutines
11. Simple Tone Subroutines
12. High Resolution Graphics
13. Additional BASIC Program Examples
### BASIC COMMANDS

Commands are executed immediately; they do not require line numbers. Most Statements (see Basic Statements Section) may also be used as commands. Remember to press Return key after each command so that Apple knows that you have finished that line. Multiple commands (as opposed to statements) on same line separated by a " : " are NOT allowed.

<table>
<thead>
<tr>
<th>COMMAND NAME</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AUTO</strong> num</td>
<td>Sets automatic line numbering mode. Starts at line number num and increments line numbers by 10. To exit AUTO mode, type a control X*, then type the letters &quot;MAN&quot; and press the return key.</td>
</tr>
<tr>
<td><strong>AUTO</strong> num1, num2</td>
<td>Same as above except increments line numbers by number num2.</td>
</tr>
<tr>
<td><strong>CLR</strong></td>
<td>Clears current BASIC variables; undimensions arrays. Program is unchanged.</td>
</tr>
<tr>
<td><strong>CON</strong></td>
<td>Continues program execution after a stop from a control C*. Does not change variables.</td>
</tr>
<tr>
<td><strong>DEL</strong> num1, num2</td>
<td>Deletes program from line number num1 through line number num2.</td>
</tr>
<tr>
<td><strong>DSP</strong> var</td>
<td>Sets debug mode that will display variable var every time that it is changed along with the line number that caused the change. (NOTE: RUN command clears DSP mode so that DSP command is effective only if program is continued by a CON or GOTO command.)</td>
</tr>
<tr>
<td><strong>HIMEM</strong> expr</td>
<td>Sets highest memory location for use by BASIC at location specified by expression expr in decimal. HIMEM: may not be increased without destroying program. HIMEM: is automatically set at maximum RAM memory when BASIC is entered by a control B*.</td>
</tr>
<tr>
<td><strong>GOTO</strong> expr</td>
<td>Causes immediate jump to line number specified by expression expr.</td>
</tr>
<tr>
<td><strong>GR</strong></td>
<td>Sets mixed color graphics display mode. Clears screen to black. Resets scrolling window. Displays 40x40 squares in 15 colors on top of screen and 4 lines of text at bottom.</td>
</tr>
<tr>
<td><strong>LIST</strong></td>
<td>Lists entire program on screen.</td>
</tr>
<tr>
<td><strong>LIST</strong> num1</td>
<td>Lists program line number num1.</td>
</tr>
<tr>
<td><strong>LIST</strong> num1, num2</td>
<td>Lists program line number num1 through line number num2.</td>
</tr>
</tbody>
</table>
LOAD exp.
Reads (Loads) a BASIC program from cassette tape.
Start tape recorder before hitting return key. Two
beeps and a ">" indicate a good load. "ERR" or "MEM"
FULL ERR" message indicates a bad tape or poor recorder
performance.

LOMEM: expr
Similar to HIMEM: except sets lowest memory location
available to BASIC. Automatically set at 2048 when
BASIC is entered with a control B*. Moving LOMEM:
destroys current variable values.

MAN
Clears AUTO line numbering mode to all manual line
numbering after a control C* or control X*.

NEW
Clears (Scratches) current BASIC program.

NO DSP var
Clears DSP mode for variable var.

NO TRACE
Clears TRACE mode.

RUN
Clears variables to zero, undimensions all arrays and
executes program starting at lowest statement line
number.

RUN expr
Clears variables and executes program starting at line
number specified by expression expr.

SAVE
Stores (saves) a BASIC program on a cassette tape.
Start tape recorder in record mode prior to hitting
return key.

TEXT
Sets all text mode. Screen is formatted to display
alpha-numeric characters on 24 lines of 40 characters
each. TEXT resets scrolling window to maximum.

TRACE
Sets debug mode that displays line number of each
statement as it is executed.

* Control characters such as control X or control C are
typed by holding down the CTRL key while typing the
specified letter. This is similar to how one holds
down the shift key to type capital letters. Control
characters are NOT displayed on the screen but are
accepted by the computer. For example, type several
control G's. We will also use a superscript C to indicate
a control character as in X^C.
### BASIC Operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Sample Statement</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prefix Operators</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( )</td>
<td>10 X = 4*(5 + X)</td>
<td>Expressions within parenthesis ( ) are always evaluated first.</td>
</tr>
<tr>
<td>+</td>
<td>20 X = 1+4*5</td>
<td>Optional; +1 times following expression.</td>
</tr>
<tr>
<td>-</td>
<td>30 ALPHA = -(BETA +2)</td>
<td>Negation of following expression.</td>
</tr>
<tr>
<td><strong>NOT</strong></td>
<td>40 IF A NOT B THEN 200</td>
<td>Logical Negation of following expression; 0 if expression is true (non-zero), 1 if expression is false (zero).</td>
</tr>
<tr>
<td><strong>Arithmetic Operators</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>↑</td>
<td>60 Y = X 3</td>
<td>Exponentiate as in X³. NOTE: ↑ is shifted letter N.</td>
</tr>
<tr>
<td>*</td>
<td>70 LET DOTS=A<em>B</em>N2</td>
<td>Multiplication. NOTE: Implied multiplication such as (2 + 3)(4) is not allowed thus N2 in example is a variable not N * 2.</td>
</tr>
<tr>
<td>/</td>
<td>80 PRINT GAMMA/S</td>
<td>Divide</td>
</tr>
<tr>
<td>MOD</td>
<td>90 X = 12 MOD 7</td>
<td>Modulo: Remainder after division of first expression by second expression.</td>
</tr>
<tr>
<td>+</td>
<td>100 X = X MOD(Y+2)</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>110 P = L + G</td>
<td>Add</td>
</tr>
<tr>
<td>=</td>
<td>120 XY4 = H-D</td>
<td>Substract</td>
</tr>
<tr>
<td></td>
<td>130 HEIGHT=15</td>
<td>Assignment operator; assigns a value to a variable. LET is optional</td>
</tr>
<tr>
<td></td>
<td>140 LET SIZE=7*5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>150 A(8) = 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>155 ALPHA$ = &quot;PLEASE&quot;</td>
<td></td>
</tr>
</tbody>
</table>
Relational and Logical Operators

The numeric values used in logical evaluation are "true" if non-zero, "false" if zero.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Sample Statement</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>160 IF D = E THEN 500</td>
<td>Expression &quot;equals&quot; expression.</td>
</tr>
<tr>
<td>=</td>
<td>170 IF A$(1,1)=&quot;Y&quot; THEN 5VV</td>
<td>String variable &quot;equal'string variable.</td>
</tr>
<tr>
<td># or &lt;&gt;</td>
<td>180 IF ALPHA #X*Y THEN 500</td>
<td>Expression &quot;does not equal&quot; expression.</td>
</tr>
<tr>
<td>#</td>
<td>190 IF A$ # &quot;NO&quot; THEN 500</td>
<td>String variable &quot;does not equal&quot; string variable. NOTE: If strings are not the same length, they are considered un-equal. &lt; &gt; not allowed with strings.</td>
</tr>
<tr>
<td>&gt;</td>
<td>200 IF A&gt;B THEN GO TO 50</td>
<td>Expression &quot;is greater than&quot; expression.</td>
</tr>
<tr>
<td>&lt;</td>
<td>210 IF A+1&lt;5 THEN 100</td>
<td>Expression &quot;is less than&quot; expression.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>220 IF A&gt;=B THEN 100</td>
<td>Expression &quot;is greater than or equal to&quot; expression.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>230 IF A+1&lt;=B-6 THEN 200</td>
<td>Expression &quot;is less than or equal to&quot; expression.</td>
</tr>
<tr>
<td>AND</td>
<td>240 IF A&gt;B AND C&lt;D THEN 200</td>
<td>Expression 1 &quot;and&quot; expression 2 must both be &quot;true&quot; for statements to be true.</td>
</tr>
<tr>
<td>OR</td>
<td>250 IF ALPHA OR BETA+1 THEN 200</td>
<td>If either expression 1 or expression 2 is &quot;true&quot;, statement is &quot;true&quot;.</td>
</tr>
</tbody>
</table>
BASIC FUNCTIONS

Functions return a numeric result. They may be used as expressions or as part of expressions. PRINT is used for examples only, other statements may be used. Expressions following function name must be enclosed between two parenthesis signs.

FUNCTION NAME

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(expr)</td>
<td>300 PRINT ABS(X)</td>
</tr>
<tr>
<td>ASC(str$)</td>
<td>310 PRINT ASC(&quot;BACK&quot;) 320 PRINT ASC(B$) 330 PRINT ASC(B$(4,4)) 335 PRINT ASC(B$(Y))</td>
</tr>
<tr>
<td>LEN(str$)</td>
<td>340 PRINT LEN(B$)</td>
</tr>
<tr>
<td>PDL(expr)</td>
<td>350 PRINT PDL(X)</td>
</tr>
<tr>
<td>PEEK(expr)</td>
<td>360 PRINT PEEK(X)</td>
</tr>
<tr>
<td>RND(expr)</td>
<td>370 PRINT RND(X)</td>
</tr>
<tr>
<td>SCR$(expr1, expr2)</td>
<td>380 PRINT SCR$(X1,Y1)</td>
</tr>
<tr>
<td>SGN(expr)</td>
<td>390 PRINT SGN(X)</td>
</tr>
</tbody>
</table>
BASIC STATEMENTS

Each BASIC statement must have a line number between 0 and 32767. Variable names must start with an alpha character and may be any number of alpha-numeric characters up to 100. Variable names may not contain buried any of the following words: AND, AT, MOD, OR, STEP, or THEN. Variable names may not begin with the letters END, LET, or REM. String variables names must end with a $ (dollar sign). Multiple statements may appear under the same line number if separated by a : (colon) as long as the total number of characters in the line (including spaces) is less than approximately 150 characters. Most statements may also be used as commands. BASIC statements are executed by RUN or GOTO commands.

NAME

CALL expr

10 CALL-936

Causes execution of a machine level language subroutine at decimal memory location specified by expression expr. Locations above 32767 are specified using negative numbers; i.e., location in example 10 is hexadecimal number $FC53.

COLOR=expr

30 COLOR=12

In standard resolution color (GR) graphics mode, this command sets screen TV color to value in expression expr in the range 0 to 15 as described in Table A. Actually expression expr may be in the range 0 to 255 without error message since it is implemented as if it were expression expr MOD 16.

DIM vari (expr1)

50 DIM A(20), B(10)

str$ (expr2)

60 DIM B$(30)

var2 (expr3)

70 DIM C (2)

Illegal:

80 DIM A(30)

Legal:

85 DIM C(1000)

The DIM statement causes APPLE II to reserve memory for the specified variables. For number arrays APPLE reserves approximately 2 times expr1 bytes of memory limited by available memory. For string arrays str$ (expr2) must be in the range of 1 to 255. Last defined variable may be redimensioned at any time; thus, example in line is illegal but 85 is allowed.

DSP var

Legal:

90 DSP AX: DSP L

Illegal:

100 DSP AX,B

102 DSP ABS

104 DSP A(5)

Legal:

105 A=A(5): DSP A

Sets debug mode that DSP variable var each time it changes and the line number where the change occurred.
<table>
<thead>
<tr>
<th>NAME</th>
<th>EXAMPLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>END</strong></td>
<td>110 END</td>
<td>Stops program execution. Sends carriage return and &quot;&gt; &quot; BASIC prompt to screen.</td>
</tr>
<tr>
<td><strong>FOR</strong> var=</td>
<td>110 FOR L=0 to 39</td>
<td>Begins FOR...NEXT loop, initializes variable var to value of expression expr1 then increments it by amount in expression expr3 each time the corresponding &quot;NEXT&quot; statement is encountered, until value of expression expr2 is reached. If STEP expr2 is omitted, a STEP of +1 is assumed. Negative numbers are allowed.</td>
</tr>
<tr>
<td><strong>STEP</strong> expr3</td>
<td>120 FOR X=Y1 TO Y3</td>
<td>130 FOR I=39 TO 1</td>
</tr>
<tr>
<td><strong>GOSUB</strong> expr</td>
<td>140 GOSUB 500</td>
<td>Causes branch to BASIC subroutine starting at legal line number specified by expression expr. Subroutines may be nested up to 16 levels.</td>
</tr>
<tr>
<td><strong>GOTO</strong> expr</td>
<td>160 GOTO 200</td>
<td>170 GOTO ALPHA+100</td>
</tr>
<tr>
<td><strong>GR</strong></td>
<td>180 GR</td>
<td>190 GR: POKE -16302,0</td>
</tr>
<tr>
<td><strong>HLIN</strong> expr1, expr2, expr3</td>
<td>200 HLIN 0,39 AT 20</td>
<td>210 HLIN 7,7+6 AT 1</td>
</tr>
</tbody>
</table>

**Note:** HLIN 0, 19 AT 0 is a horizontal line at the top of the screen extending from left corner to center of screen and HLIN 29, 39 AT 39 is a horizontal line at the bottom of the screen extending from center to right corner.
IF expression 220 IF A > B THEN  
THEN statement       PRINT A  
230 IF X = 0 THEN C = 1  
240 IF A ≠ 10 THEN  
GOSUB 200  
250 IF A$(1,1) = "Y" THEN 100  
Illegal:  
260 IF L > 5 THEN 50: ELSE 60  
Legal:  
270 IF L > 5 THEN 50  
GO TO 60  

INPUT var1, var2, str$ 280 INPUT X, Y, Z(3)  
290 INPUT "AMT",  
      DLLR  
300 INPUT "Y or N?", A$  

En ters data into memory from I/O  
device. If number input is expected,  
APPLE will output "?": if string input is  
expected no "?" will be outputted. Multiple  
numeric inputs to same statement may be  
separated by a comma or a carriage return.  
String inputs must be separated by a  
carriage return only. One pair of " " may  
be used immediately after INPUT to output  
prompting text enclosed within the quotation  
marks to the screen.  

IN# expr 310 IN# 6  
320 IN# Y+2  
330 IN# 0  

Transfers source of data for subsequent  
INPUT statements to peripheral I/O slot  
(1-7) as specified as by expression expr.  
Slot 0 is not addressable from BASIC.  
IN#0 (Example 330) is used to return data  
source from peripheral I/O to keyboard  
connector.  

LET 340 LET X = 5  

Assignment operator. "LET" is optional  

LIST num1,       350 IF X > 6 THEN LIST 50  
num2  

Causes program from line number num1  
through line number num2 to be displayed  
on screen.  

NEXT var1, var2 360 NEXT I  
370 NEXT J, K  

Increments corresponding "FOR" variable  
and loops back to statement following  
"FOR" until variable exceeds limit.  

NO DSP var 380 NO DSP I  

Turns-off DSP debug mode for variable  

NO TRACE 390 NO TRACE  

Turns-off TRACE debug mode
PLOT, expr1, expr2
400 PLOT 15, 25
400 PLOT XV, YV
In standard resolution color
graphics, this command plots a small
square of a predefined color (set
by COLOR=) at horizontal location
specified by expression expr1 in
range 0 to 39 and vertical location
specified by expression expr2 in range
0 to 39 (or 0 to 47 if in all graphics
mode) NOTE: PLOT 0 0 is upper left
and PLOT 39, 39 (or PLOT 39, 47) is
lower right corner.

POKE expr1, expr2
420 POKE 20, 40
430 POKE 7*256, XMOD255
Stores decimal number defined by
expression expr2 in range of 0
255 at decimal memory location
specified by expression expr1.
Locations above 32767 are specified
by negative numbers.

POP
440 POP
"POPS" nested GOSUB return stack
address by one.

PRINT var1, var, str$ 
450 PRINT L1
460 PRINT L1, X2
470 PRINT "AMT=";DX
480 PRINT A$;B$;
490 PRINT
492 PRINT "HELLO"
494 PRINT 2+3
Outputs data specified by variable
var or string variable str$ starting
at current cursor location. If there
is not trailing "," or ";" (Ex 450)
a carriage return will be generated.
Commas (Ex. 460) outputs data in 5
left justified columns. Semi-colon
(Ex. 470) inhibits print of any spaces.
Text imbedded in " " will be printed
and may appear multiple times.

PR# expr
500 PR# 7
Like IN#, transfers output to I/O
slot defined by expression expr. PR#
0 is video output not I/O slot 0.

REM
510 REM REMARK
No action. All characters after REM
are treated as a remark until terminated
by a carriage return.

RETURN
520 RETURN
530 IF X= 5 THEN
RETURN
Causes branch to statement following
last GOSUB; i.e., RETURN ends a
subroutine. Do not confuse "RETURN"
statement with Return key on keyboard.
**TAB expr**

53Ø TAB 24
54Ø TAB I+24
55Ø IF A#B THEN
   TAB 20

Moves cursor to absolute horizontal position specified by expression `expr` in the range of 1 to 40. Position is left to right.

**TEXT**

55Ø TEXT
56Ø TEXT: CALL-936

Sets all text mode. Resets scrolling window to 24 lines by 40 characters. Example 56Ø also clears screen and homes cursor to upper left corner.

**TRACE**

57Ø TRACE
580 IFN > 32ØØØ
   THEN TRACE

Sets debug mode that displays each line number as it is executed.

**VLIN expr1, expr2**

59Ø VLIN ø, 39AT15
5Ø0 VLIN Z,Z+6ATY

Similar to HLIN except draws vertical line starting at `expr1` and ending at `expr2` at horizontal position `expr3`.

**VTAB expr**

61Ø VTAB 18
62Ø VTAB Z+2

Similar to TAB. Moves cursor to absolute vertical position specified by expression `expr` in the range 1 to 24. VTAB 1 is top line on screen; VTAB 24 is bottom.
SPECIAL CONTROL AND EDITING CHARACTERS

"Control" characters are indicated by a super-scripted "C" such as G\textsuperscript{C}. They are obtained by holding down the CTRL key while typing the letter. Control characters are NOT displayed on the TV screen. B and C must be followed by a carriage return. Screen editing characters are indicated by a sub-scripted "E" such as D\textsubscript{E}. They are obtained by pressing and releasing the ESC key then typing specified letter. Edit characters send information only to display screen and does not send data to memory. For example, U\textsuperscript{C} moves to cursor to right and copies text while A\textsubscript{E} moves to cursor to right but does not copy text.

<table>
<thead>
<tr>
<th>CHARACTER</th>
<th>DESCRIPTION OF ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET key</td>
<td>Immediately interrupts any program execution and resets computer. Also sets all text mode with scrolling window at maximum. Control is transferred to System Monitor and Apple prompts with a &quot;**&quot; (asterisk) and a bell. Hitting RESET key does NOT destroy existing BASIC or machine language program.</td>
</tr>
<tr>
<td>Control B</td>
<td>If in System Monitor (as indicated by a &quot;**&quot;), a control B and a carriage return will transfer control to BASIC, scratching (killing) any existing BASIC program and set HIMEM: to maximum installed user memory and LOMEM: to 2048.</td>
</tr>
<tr>
<td>Control C</td>
<td>If in BASIC, halts program and displays line number where stop occurred*. Program may be continued with a CON command. If in System Monitor, (as indicated by &quot;**&quot;), control C and a carriage return will enter BASIC without killing current program.</td>
</tr>
<tr>
<td>Control G</td>
<td>Sounds bell (beeps speaker)</td>
</tr>
<tr>
<td>Control H</td>
<td>Backspaces cursor and deletes any overwritten characters from computer but not from screen. Apply supplied keyboards have special key &quot;±&quot; on right side of keyboard that provides this functions without using control button.</td>
</tr>
<tr>
<td>Control 3</td>
<td>Issues line feed only</td>
</tr>
<tr>
<td>Control V</td>
<td>Compliment to H\textsuperscript{C}. Forward spaces cursor and copies over written characters. Apple keyboards have H-0 key on right side which also performs this function.</td>
</tr>
<tr>
<td>Control X</td>
<td>Immediately deletes current line.</td>
</tr>
</tbody>
</table>

* If BASIC program is expecting keyboard input, you will have to hit carriage return key after typing control C.
<table>
<thead>
<tr>
<th>CHARACTER</th>
<th>DESCRIPTION OF ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Move cursor to right</td>
</tr>
<tr>
<td></td>
<td>Move cursor to left</td>
</tr>
<tr>
<td></td>
<td>Move cursor down</td>
</tr>
<tr>
<td></td>
<td>Move cursor up</td>
</tr>
<tr>
<td></td>
<td>Clear text from cursor to end of line</td>
</tr>
<tr>
<td></td>
<td>Clear text from cursor to end of page</td>
</tr>
<tr>
<td>@</td>
<td>Home cursor to top of page, clear text to end of page.</td>
</tr>
</tbody>
</table>

Table A:  APPLE II COLORS AS SET BY COLOR –

Note: Colors may vary depending on TV tint (hue) setting and may also be changes by adjusting trimmer capacitor C3 on APPLE II P.C. Board.

<table>
<thead>
<tr>
<th>COLOR</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>1</td>
<td>Magenta</td>
</tr>
<tr>
<td>2</td>
<td>Bark Blue</td>
</tr>
<tr>
<td>3</td>
<td>Light Purple</td>
</tr>
<tr>
<td>4</td>
<td>Dark Green</td>
</tr>
<tr>
<td>5</td>
<td>Grey</td>
</tr>
<tr>
<td>6</td>
<td>Medium Blue</td>
</tr>
<tr>
<td>7</td>
<td>Light Blue</td>
</tr>
<tr>
<td>8</td>
<td>Brown</td>
</tr>
<tr>
<td>9</td>
<td>Orange</td>
</tr>
<tr>
<td>10</td>
<td>Grey</td>
</tr>
<tr>
<td>11</td>
<td>Pink</td>
</tr>
<tr>
<td>12</td>
<td>Green</td>
</tr>
<tr>
<td>13</td>
<td>Yellow</td>
</tr>
<tr>
<td>14</td>
<td>Blue/Green</td>
</tr>
<tr>
<td>15</td>
<td>White</td>
</tr>
</tbody>
</table>
### Special Controls and Features

#### Hex | BASIC Example | Description
--- | --- | ---

**Display Mode Controls**

- **C050** 10 POKE -16304,0  
  Set color graphics mode
- **C051** 20 POKE -16303,0  
  Set text mode
- **C052** 30 POKE -16302,0  
  Clear mixed graphics
- **C053** 40 POKE -16301,0  
  Set mixed graphics (4 lines text)
- **C054** 50 POKE -16300,0  
  Clear display Page. 2 (BASIC commands use Page 1 only)
- **C055** 60 POKE -16299,0  
  Set display to Page 2 (alternate)
- **C056** 70 POKE -16298,0  
  Clear HIRES graphics mode
- **C057** 80 POKE -16297,0  
  Set HIRES graphics mode

**TEXT Mode Controls**

- **C0E0** 90 POKE 32,L1  
  Set left side of scrolling window to location specified by L1 in range of 0 to 39.
- **C0E1** 100 POKE 33,W1  
  Set window width to amount specified by W1. L1+W1<40. W1>0
- **C0E2** 110 POKE 34,11  
  Set window top to line specified by T1 in range of 0 to 23
- **C0E3** 120 POKE 35,B1  
  Set window bottom to line specified by B1 in the range of 0 to 23. B1>T1
- **C0E4** 130 CH=PEEK(36)  
  Read/set cusor horizontal position in the range of 0 to 39. If using TAB, you must add "1" to cusor position read value; Ex. 140 and 150 perform identical function.
- **C0E5** 140 POKE 36,CH  
  150 TAB(CH+1)
- **C0E6** 160 CV=PEEK (37)  
  170 POKE 37,CV  
  180 VTAB(CV+1)  
  Similar to above. Read/set cusor vertical position in the range 0 to 23.
- **C0E7** 190 POKE 50,127  
  Set inverse flag if 127 (Ex. 190)
- **C0E8** 200 POKE 50,255  
  Set normal flag if 255(Ex. 200)
- **FC58** 210 CALL -936  
  (@E) Home cusor, clear screen
- **FC42** 220 CALL -958  
  (FE) Clear from cusor to end of page
<table>
<thead>
<tr>
<th>Hex</th>
<th>BASIC Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC9C</td>
<td>230 CALL -868</td>
<td>(EE) Clear from cursor to end of line</td>
</tr>
<tr>
<td>FC66</td>
<td>240 CALL -922</td>
<td>(JC) Line feed</td>
</tr>
<tr>
<td>FC70</td>
<td>250 CALL -912</td>
<td>Scroll up text one line</td>
</tr>
</tbody>
</table>

**Miscellaneous**

<table>
<thead>
<tr>
<th>Hex</th>
<th>BASIC Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C030</td>
<td>360 X=PEEK(-16336) 365 POKE -16336,0</td>
<td>Toggle speaker</td>
</tr>
<tr>
<td>C000</td>
<td>370 X=PEEK(-16384)</td>
<td>Read keyboard; if X&gt;127 then key was pressed.</td>
</tr>
<tr>
<td>C010</td>
<td>380 POKE -16368,0</td>
<td>Clear keyboard strobe - always after reading keyboard.</td>
</tr>
<tr>
<td>C061</td>
<td>390 X=PEEK(16287)</td>
<td>Read PDL(0) push button switch. If X&gt;127 then switch is &quot;on&quot;.</td>
</tr>
<tr>
<td>C062</td>
<td>400 X=PEEK(-16286)</td>
<td>Read PDL(1) push button switch.</td>
</tr>
<tr>
<td>C063</td>
<td>410 X=PEEK(-16285)</td>
<td>Read PDL(2) push button switch.</td>
</tr>
<tr>
<td>C058</td>
<td>420 POKE -16296,0</td>
<td>Clear Game I/O AN0 output</td>
</tr>
<tr>
<td>C059</td>
<td>430 POKE -16295,0</td>
<td>Set Game I/O AN0 output</td>
</tr>
<tr>
<td>C05A</td>
<td>440 POKE -16294,0</td>
<td>Clear Game I/O AN1 output</td>
</tr>
<tr>
<td>C05B</td>
<td>450 POKE -16293,0</td>
<td>Set Game I/O AN1 output</td>
</tr>
<tr>
<td>C05C</td>
<td>460 POKE -16292,0</td>
<td>Clear Game I/O AN2 output</td>
</tr>
<tr>
<td>C05D</td>
<td>470 POKE -16291,0</td>
<td>Set Game I/O AN2 output</td>
</tr>
<tr>
<td>C05E</td>
<td>480 POKE -16290,0</td>
<td>Clear Game I/O AN3 output</td>
</tr>
<tr>
<td>C05F</td>
<td>490 POKE -16289,0</td>
<td>Set Game I/O AN3 output</td>
</tr>
</tbody>
</table>
### APPLE II BASIC ERROR MESSAGES

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*** SYNTAX ERR</td>
<td>Results from a syntactic or typing error.</td>
</tr>
<tr>
<td>*** &gt; 32767 ERR</td>
<td>A value entered or calculated was less than -32767 or greater than 32767.</td>
</tr>
<tr>
<td>*** &gt; 255 ERR</td>
<td>A value restricted to the range 0 to 255 was outside that range.</td>
</tr>
<tr>
<td>*** BAD BRANCH ERR</td>
<td>Results from an attempt to branch to a non-existent line number.</td>
</tr>
<tr>
<td>*** BAD RETURN ERR</td>
<td>Results from an attempt to execute more RETURNs than previously executed GOSUBs.</td>
</tr>
<tr>
<td>*** BAD NEXT ERR</td>
<td>Results from an attempt to execute a NEXT statement for which there was not a corresponding FOR statement.</td>
</tr>
<tr>
<td>*** 16 GOSUBS ERR</td>
<td>Results from more than 16 nested GOSUBs.</td>
</tr>
<tr>
<td>*** 16 FORS ERR</td>
<td>Results from more than 16 nested FOR loops.</td>
</tr>
<tr>
<td>*** NO END ERR</td>
<td>The last statement executed was not an END.</td>
</tr>
<tr>
<td>*** MEM FULL ERR</td>
<td>The memory needed for the program has exceeded the memory size allotted.</td>
</tr>
<tr>
<td>*** TOO LONG ERR</td>
<td>Results from more than 12 nested parentheses or more than 128 characters in input line.</td>
</tr>
<tr>
<td>*** DIM ERR</td>
<td>Results from an attempt to DIMension a string array which has been previously dimensioned.</td>
</tr>
<tr>
<td>*** RANGE ERR</td>
<td>An array was larger than the DIMensioned value or smaller than 1 or HLIN, VLIN, PLOT, TAB, or VTAB arguments are out of range.</td>
</tr>
<tr>
<td>*** STR OVFL ERR</td>
<td>The number of characters assigned to a string exceeded the DIMensioned value for that string.</td>
</tr>
<tr>
<td>*** STRING ERR</td>
<td>Results from an attempt to execute an illegal string operation.</td>
</tr>
<tr>
<td>RETYPE LINE</td>
<td>Results from illegal data being typed in response to an INPUT statement. This message also requests that the illegal item be retyped.</td>
</tr>
</tbody>
</table>
Simplified Memory Map

FFFF 64K Monitor and BASIC Routines in ROM
E000 56K Future enhancement or user supplied PROMS
D000 52K Peripheral I/O
C600 48K

XX XX - - User specified RAM memory size (HIMEM:)

XX (LOMEM:)

7FF 2K Screen Memory
400 1K
0 0 Internal Workspace

User Workspace
READ/SAVE DATA SUBROUTINE

INTRODUCTION

Valuable data can be generated on the Apple II computer and sometimes it is useful to have a software routine that will allow making a permanent record of this information. This paper discusses a simple subroutine that serves this purpose.

Before discussing the Read/Save routines a rudimentary knowledge of how variables are mapped into memory is needed.

Numeric variables are mapped into memory with four attributes. Appearing in order sequentially are the Variable Name, the Display Byte, the Next Variable Address, and the Data of the Variable. Diagramatically this is represented as:

\[
\begin{array}{cccc}
\text{YN} & \text{DSP} & \text{NVA} & \text{DATA}(0) \quad \text{DATA}(1) \quad \ldots \quad \text{DATA}(N) \\
1 & h_1 & h_2 & h_{n+1}
\end{array}
\]

- **VARIABLE NAME** - up to 100 characters represented in memory as ASCII equivalents with the high order bit set.

- **DSP (DISPLAY) BYTE** - set to 01 when DSP set in BASIC initiates a process that displays this variable with the line number every time it is changed within a program.

- **NVA (NEXT VARIABLE ADDRESS)** - two bytes (first low order, the second high order) indicating the memory location of the next variable.

- **DATA** - hexadecimal equivalent of numeric information, represented in pairs of bytes, low order byte first.
String variables are formatted a bit differently than numeric ones. These variables have one extra attribute - a string terminator which designates the end of a string. A string variable is formatted as follows:

\[
\begin{array}{cccccccc}
\text{VN} & \text{DSP} & \text{NVA} & \text{DATA}(0) & \text{DATA}(1) & \ldots & \text{DATA}(n) & \text{ST} \\
1 & h_1 & h_2 & h_{n+1} & & & & \\
\end{array}
\]

- **VARIABLE NAME** - up to 100 characters represented in memory as ASCII equivalents with the high order bit set.
- **DSP (DISPLAY) BYTE** - set to 01 when DSP set in BASIC, initiates a process that displays this variable with the line number every time it is changed within a program.
- **NVA (NEXT VARIABLE ADDRESS)** - two bytes (first low order, the second high order) indicating the memory location of the next variable.
- **DATA** - ASCII equivalents with high order bit set.
- **STRING TERMINATOR (ST)** - none high order bit set character indicating END of string.

There are two parts of any BASIC program represented in memory. One is the location of the variables used for the program, and the other is the actual BASIC program statements. As it turns out, the mapping of these within memory is a straightforward process. Program statements are placed into memory starting at the top of RAM memory* unless manually shifted by the "HIMEM:" command, and are pushed down as each new (numerically larger) line numbered statement is entered into the system. Figure 1a illustrates this process diagramatically. Variables on the other hand are mapped into memory starting at the lowest position of RAM memory - hex $8000$ (2048) unless manually shifted by the"LOMEM:" command. They are laid down from there (see Figure 1b) and continue until all the variables have been mapped into memory or until they collide with the program statements. In the event of the latter case a memory full error will be generated.

*Top of RAM memory is a function of the amount of memory. 16384 will be the value of "HIMEM:" for a 16K system.
The computer keeps track of the amount of memory used for the variable
and program statements. By placing the end memory location of each into
$CC$-$CD(204-205)$ and $CA$-$CB(203-204)$, respectively. These are the BASIC
memory program pointers and their values can be found by using the statements
in Figure 2. CM defined in Figure 1 as the location of the end of the variable
tape is equal to the number resulting from statement a of Figure 2. PP, the
program pointer, is equal to the value resulting from statement 2b. These
statements (Figure 2) can then be used on any Apple II computer to find the
limits of the program and variable table.

FINDING THE VARIABLE TABLE FROM BASIC

First, power up the Apple II, reset it, and use the CTRL B (control B)
command to place the system into BASIC initializing the memory pointers. Using
the statements from Figure 2 it is found that for a 16K Apple II CM is equal to
2048 and PP is equal to 16384. These also happen to be the values of OMEN and
HIMEN: But this is expected because upon using the Bc command both memory
pointers are initialized indicating no program statements and no variables.

To illustrate what a variable table looks like in Apple II memory suppose
we want to assign the numeric variable A ($C1$ is the ASCII equivalent of a with
the high order bit set) the value of -1 (FF FF in hex) and then examine the
memory contents. The steps in this process are outlined in example I. Variable A
is defined as equal to -1 (step 1). Then for convenience another variable - B -
is defined as equal to 0 (step 2). Now that the variable table has been defined
use of statement 2a indicates that CM is equal to 2060 (step 3). LOMEN has not
been readjusted so it is equal to 2048. Therefore the variable table resides in
memory from 2048 ($800$ hex) to 2060 ($88C$). Depressing the "RESET" key places
the Apple II into the monitor mode (step 4).

We are now ready to examine the memory contents of the variable table.
Since the variable table resides from $800$ hex to $80C$ hex typing in "800.80C"
and then depressing the "RETURN" key (step 5) will list the memory contents of
this range. Figure 3 lists the contents with each memory location labelled.
Examining these contents we see that C1 is equal to the variable name and is the
memory equivalent of "A" and that FF FF is the equivalent of -1. From this, since
the variable name is at the beginning of the table and the data is at the end, the
variable table representation of A extends from $800$ to $805$. We have then found
the memory range of where the variable A is mapped into memory.
The reason for this will become clear in the next section.

READ/SAVE ROUTINE

The READ/SAVE subroutine has three parts. The first section (lines 0-10) defines variable A and transfers control to the main program. Lines 20 through 26 represents the Write data to tape routine and lines 30-38 represent the Read data from tape subroutine. Both READ and SAVE routines are executable by the BASIC "GOSUB X" (where X is 20 for write and 30 is for read) command. And as listed these routines can be directly incorporated into almost any BASIC program for read and saving a variable table. The limitation of these routines is that the whole part of a variable table is processed so it is necessary to maintain exactly the dimension statements for the variables used.

The variables used in this subroutine are defined as follows:

A = record length, must be the first variable defined
CM= the value obtained from statement a of figure 2
LW= is equal to the value of "LOMEM:"
    Nominally 2048

SAVING A DATA TABLE

The first step in a hard copy routine is to place the desired data onto tape. This is accomplished by determining the length of the variable table and setting A equal to it. Next within the main program when it is time to write the data a GOSUB20 statement will execute the write to tape process. Record length, variable A, is written to tape first (line 22) followed by the desired data (line 24). When this process is completed control is returned to the main program.

READING A DATA TABLE

The second step is to read the data from tape. When it is time a GOSUB30 statement will initiate the read process. First, the record length is read in and checked to see if enough memory is available (line 32-34). If exactly the same dimension statements are used it is almost guaranteed that there will be enough memory available. After this the variable table is read in (line 34) and control is then returned to the main program (line 36). If not enough memory is available then an error is generated and control is returned to the main program (line 38)
EXAMPLE OF READ/SAVE USAGE

The Read/Save routines may be incorporated directly into a main program. To illustrate this a test program is listed in example 2. This program dimensions a variable array of twenty by one, fills the array with numbers, writes the data table to tape, and then reads the data from tape listing the data on the video display. To get a feeling for how to use these routines enter this program and explore how the Read/Save routines work.

CONCLUSION

Reading and Saving data in the format of a variable table is a relatively straight forward process with the Read/Save subroutine listed in figure 4. This routine will increase the flexibility of the Apple II by providing a permanent record of the data generated within a program. This program can be reprocessed. The Read/Save routines are a valuable addition to any data processing program.
Figure 1

a) PRINT PEEK(204) + PEEK(205)*256 → PP

b) PRINT PEEK(202) + PEEK(203)*256 → CM

Figure 2

Figure 3

$800.80C rewritten with labelling
READ/SAVE PROGRAM

Ø  A=Ø

1Ø  GOTO 1ØØ

2Ø  PRINT "REWIND TAPE THEN
START TAPE RECORDER":
INPUT "THEN HIT RETURN",
B$

22  A=CM-LM: POKE 6Ø,4:
POKE 61,8: POKE 62,5:
POKE 63,8: CALL -3Ø7

24  POKE 6Ø,LM MOD 256:
POKE 61, LM/256:
POKE 62, CM MOD 256:
POKE 63, CM/256:
CALL -3Ø7

26  PRINT "DATA TABLE SAVED":
RETURN

3Ø  PRINT "REWIND THE TAPE
THEN START TAPE RECORDER":
INPUT "AND HIT RETURN",
B$

32  POKE 6Ø,4: POKE 61,8:
POKE 62,5: POKE 63,8:
CALL -259

34  IF A<01 THEN 38: P=LM+A:
IF P>HM THEN 38: CM=P:
POKE 6Ø, LM MOD 256:
POKE 61, LM/256: POKE 52,
CM MOD 256: POKE 63, CM/256:
CALL -259

36  PRINT "DATA READ IN":
RETURN

38  PRINT "***TOO MUCH DATA
BASE***": RETURN

NOTE: CM, LM and A must be defined within the main program.
Define variable A=-1, then hit RETURN

Define variable B=Ø, then hit RETURN

Use statement 2a to find the end of the VARIABLE TABLE

Hit the RESET key, Apple moves into Monitor mode.

Type in VARIABLE TABLE RANGE and HIT the RETURN KEY.

Example 1
Example 2

```
10 REM WRITE DATA TO TAPE ROUTINE
20 A=CHR(1); B=68; C: POKE 68,A: POKE 61,61,B: CALL -387
22 POKE 60,LC MOD 256: POKE 61,LC
24 POKE 62,LC CH MOD 256 : POKE 63,CH CH MOD 256
26 RETURN

30 REM READ DATA SUBROUTINE
32 POKE 60,A: POKE 61,B: POKE 62,A: CALL -259
34 IF A>0 THEN 36: IF A>8 THEN 36: IF A>0 THEN 36
36 POKE 60,LC: POKE 61,LC
38 RETURN

40 PRINT "*** ALL NUMBERS GENERATED***"
50 CALL -386: PRINT "YOU ARE ABOUT TO WRITE OUT THE DATA";
60 PRINT "IF YOU ARE READY START THE RECORDER IN RECORD MODE";
70 PRINT "AND HIT RETURN"
80 CALL -326: PRINT "YOU ARE ABOUT TO WRITE OUT THE DATA TO TAPE";
90 PRINT "YOU ARE ABOUT TO CLEAR THE X:203 TABLE AND READ THE DATA FROM TAPE"
100 FOR I=1 TO 20: X(I)=I: NEXT I
110 PRINT "YOU ARE ABOUT TO CLEAR THE X:203 TABLE AND READ THE DATA FROM TAPE"
120 FOR I=1 TO 20: X(I)=I: NEXT I
130 PRINT "YOU ARE ABOUT TO CLEAR THE X:203 TABLE AND READ THE DATA FROM TAPE"
140 FOR I=1 TO 20: X(I)=I: NEXT I
150 PRINT "YOU ARE ABOUT TO CLEAR THE X:203 TABLE AND READ THE DATA FROM TAPE"
160 FOR I=1 TO 20: X(I)=I: NEXT I
170 PRINT "YOU ARE ABOUT TO CLEAR THE X:203 TABLE AND READ THE DATA FROM TAPE"
180 FOR I=1 TO 20: X(I)=I: NEXT I
190 FOR I=1 TO 20: X(I)=I: NEXT I
200 END
```
INTRODUCTION
Computers can perform marvelous feats of mathematical computation at well beyond the speed capable of most human minds. They are fast, cold and accurate; man on the other hand is slower, has emotion, and makes errors. These differences create problems when the two interact with one another. So to reduce this problem humanizing of the computer is needed. Humanizing means incorporating within the computer procedures that aid in a program's usage. One such technique is the addition of a tone subroutine. This paper discusses the incorporation and usage of a tone subroutine within the Apple II computer.

Tone Generation
To generate tones in a computer three things are needed: a speaker, a circuit to drive the speaker, and a means of triggering the circuit. As it happens the Apple II computer was designed with a two-inch speaker and an efficient speaker driving circuit. Control of the speaker is accomplished through software.

Toggling the speaker is a simple process, a mere PEEK - 16336 ($C030) in BASIC statement will perform this operation. This does not, however, produce tones, it only emits clicks. Generation of tones is the goal, so describing frequency and duration is needed. This is accomplished by toggling the speaker at regular intervals for a fixed period of time. Figure 1 lists a machine language routine that satisfies these requirements.

Machine Language Program
This machine language program resides in page 0 of memory from $92 (2) to $14 (20). $00 (00) is used to store the relative period (P) between toggling of the speaker and $01 (01) is used as the memory location for the value of relative duration (0). Both P and D can range in value from $00 (0) to $FF (255). After the values for frequency and duration are placed into memory a CALL2 statement from BASIC will activate this routine. The speaker is toggled with the machine language statement residing at $02 and then a
delay in time equal to the value in $00$ occurs. This process is repeated until
the tone has lasted a relative period of time equal to the duration (value in $01$)
and then this program is exited (statement $14$).

Basic Program

The purpose of the machine language routine is to generate tones controllable
from BASIC as the program dictates. Figure 2 lists the appropriate statement that
will deposit the machine language routine into memory. They are in the form of
a subroutine and can be activated by a GOSUB 32000 statement. It is only necessary
to use this statement once at the beginning of a program. After that the machine
language program will remain in memory unless a later part of the main program
modifies the first 20 locations of page 0.

After the GOSUB 32000 has placed the machine language program into memory
it may be activated by the statement in Figure 3. This statement is also in the
form of a GOSUB because it can be used repetitively in a program. Once the frequency
and duration have been defined by setting P and D equal to a value between
0 and 255 a GOSUB 25 statement is used to initiate the generation of a tone. The
values of P and D are placed into $00$ and $01$ and the CALL2 command activates the
machine language program that toggles the speaker. After the tone has ended
control is returned to the main program.

The statements in Figures 2 and 3 can be directly incorporated into BASIC
programs to provide for the generation of tones. Once added to a program an
infinite variety of tone combinations can be produced. For example, tones can
be used to prompt, indicate an error in entering or answering questions, and
supplement video displays on the Apple II computer system.

Since the computer operates at a faster rate than man does, prompting can
be used to indicate when the computer expects data to be entered. Tones can be
generated at just about any time for any reason in a program. The programmer’s
imagination can guide the placement of these tones.

CONCLUSION

The incorporation of tones through the routines discussed in this paper
will aid in the humanizing of software used in the Apple computer. These routines
can also help in transforming a dull program into a lively one. They are relatively
easy to use and are a valuable addition to any program.
FIGURE 1. Machine Language Program
adapted from a program by P. Lutas.

32000 POKE 2,173: POKE 3,46: POKE
4,192: POKE 5,136: POKE 6,208
8,74: POKE 8,156: POKE 9,256
10,246: POKE 11,8: POKE 12,1024: POKE
13,246: POKE 14,246: POKE 15
16,4: POKE 16,4: POKE 17,76
18,2: POKE 19,8: POKE
20,96: RETURN

FIGURE 2. BASIC "POKES"

C5 POKE 8,F: POKE 1,0: CALL 2:
RETURN

FIGURE 3. GOSUB
High-Resolution Operating Subroutines

These subroutines were created to make programming for High-Resolution Graphics easier, for both BASIC and machine-language programs. These subroutines occupy 757 bytes of memory and are available on either cassette tape or Read-Only Memory (ROM). This note describes use and care of these subroutines.

There are seven subroutines in this package. With these, a programmer can initialize High-Resolution mode, clear the screen, plot a point, draw a line, or draw and animate a predefined shape on the screen. There are also some other general-purpose subroutines to shorten and simplify programming.

BASIC programs can access these subroutines by use of the CALL statement, and can pass information by using the POKE statement. There are special entry points for most of the subroutines that will perform the same functions as the original subroutines without modifying any BASIC pointers or registers. For machine language programming, a JSR to the appropriate subroutine address will perform the same function as a BASIC CALL.

In the following subroutine descriptions, all addresses given will be in decimal. The hexadecimal substitutes will be preceded by a dollar sign ($). All entry points given are for the cassette tape subroutines, which load into addresses C00 to FFF (hex). Equivalent addresses for the ROM subroutines will be in italic type face.
High-Resolution Operating Subroutines

**INIT** Initializes High-Resolution Graphics mode.

From BASIC: CALL 3972 (or CALL -12288)

From machine language: JSR $C00 (or JSR $D000)

This subroutine sets High-Resolution Graphics mode with a 280 x 160 matrix of dots in the top portion of the screen and four lines of text in the bottom portion of the screen. INIT also clears the screen.

**CLEAR** Clears the screen.

From BASIC: CALL 3086 (or CALL -12274)

From machine language: JSR $C0E (or JSR $D00E)

This subroutine clears the High-Resolution screen without resetting the High-Resolution Graphics mode.

**PLOT** Plots a point on the screen.

From BASIC: CALL 3780 (or CALL -11580)

From machine language: JSR $C7C (or JSR $D07C)

This subroutine plots a single point on the screen. The X and Y coordinates of the point are passed in locations 800, 801, and 802 from BASIC, or in the A, X, and Y registers from machine language. The Y (vertical) coordinate can be from 0
High-Resolution Operating Subroutines

PLOT (continued)

(top of screen) to 159 (bottom of screen) and is passed in
location 882 or the A-register; but the X (horizontal) coordinate
can range from 0 (left side of screen) to 279 (right side of screen)
and must be split between locations 880 (X MOD 256) and 881
(X/256). or, from machine language, between registers X (X LO)
and Y (X HI). The color of the point to be plotted must be set
in location 812 ($32C). Four colors are possible: 0 is BLACK,
85 ($55) is GREEN, 170 ($AA) is VIOLET, and 255 ($FF) is WHITE.

POSN Positions a point on the screen.

From BASIC: CALL 3761 (or CALL -11599)

From machine language: JSR $C26 (or JSR $D926)

This subroutine does all calculations for a PLOT, but does
not plot a point (it leaves the screen unchanged). This is useful
when used in conjunction with LINE or SHAPE (described later).
To use this subroutine, set up the X and Y coordinates just the:
same as for PLOT. The color in location 812 ($32C) is ignored.

LINE Draw a line on the screen.
High-Resolution Operating Routines

LINE Draws a line on the screen.
From BASIC: CALL 3736 (or CALL -11574)
From machine language: JSR $C95 (or JSR $0995)

This subroutine draws a line from the last point PLOTted or POSN'ed to the point specified. One endpoint is the last point PLOTted or POSN'ed; the other endpoint is passed in the same manner as for a PLOT or POSN. The color of the line is set in location 812 ($32C). After the line is drawn, the new endpoint becomes the base endpoint for the next line drawn.

SHAPE Draws a predefined shape on the screen.
From BASIC: CALL 3885 (or CALL -11555)
From machine language: JSR $DBC (or JSR $D1BC)

This subroutine draws a predefined shape on the screen at the point previously PLOTted or POSN'ed. The shape is defined by a table of vectors in memory. (How to create a vector table will be described later). The starting address of this table should be passed in locations 804 and 805 from BASIC or in the Y and X registers from machine language. The color of the shape should be passed in location 28 ($1C).

There are two special variables that are used only with shapes: the scaling factor and the rotation factor. The scaling factor determines the relative size of the shape. A scaling factor of
SHAPE (continued)

I will cause the shape to be drawn true size, while a scaling
factor of 2 will draw the shape double size, etc. The scaling
factor is passed in location 896 from BASIC or $32F from machine
language. The rotation factor specifies one of 64 possible angles
of rotation for the shape. A rotation factor of $0 will cause the
shape to be drawn right-side up, where a rotation factor if 16
will draw the shape rotated 90° clockwise, etc. The rotation
factor is passed in location 897 from BASIC of in the A-register
from machine language.

The table of vectors which defines the shape to be drawn is
a series of bytes stored in memory. Each byte is divided into
three sections, and each section specifies whether or not to plot
a point and also a direction to move (up, down, left, or right).
The SHAPE subroutine steps through the vector table byte by byte,
and then through each byte section by section. When it reaches
a $0 byte, it is finished.

The three sections are arranged in a byte like this:

```
    7   6   5   4   3   2   1   0
   DD PP DD PP DD PP DD PP
```

Each bit pair DD specifies a direction to move, and the two bits
P specify whether or not to plot a point before moving. Notice
that the last section (most significant bits) does not have a P
field, so it can only be a move without plotting. The SHAPE
SHAPE (continued)

subroutine processes the sections from right to left (least significant bit to most significant bit). IF THE REMAINING SECTIONS OF THE BYTE ARE ZERO, THEN THEY ARE IGNORED. Thus, the byte cannot end with sections of $\emptyset$ (move up without plotting).

Here is an example of how to create a vector table:

Suppose we want to draw a shape like this:

First, draw it on graph paper, one dot per square. Then decide where to start drawing the shape. Let's start this one in the center. Next, we must draw a path through each point in the shape, using only $90^\circ$ angles on the turns:

Next, re-draw the shape as a series of vectors, each one moving one place up, down, left, or right, and distinguish the vectors that plot a point before moving:

Now "unwrap" those vectors and write them in a straight line:

Now draw a table like the one in Figure 1. For each vector in the line, figure the bit code and place it in the next available section in the table. If it will not fit or is a $\emptyset$ at the end of a byte, then skip that section and go on to the next. When you have finished
coding all vectors, check your work to make sure it is accurate. Then make another table (as in figure 2) and re-copy the coded vectors from the first table. Then decode the vector information into a series of hexadecimal bytes, using the hexadecimal code table in figure 3. This series of hexadecimal bytes is your shape definition table, which you can now put into the Apple II's memory and use to draw that shape on the screen.
Shape vectors: $\downarrow \downarrow \leftarrow \leftarrow \uparrow \uparrow \uparrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \downarrow \downarrow \leftarrow \leftarrow$

**Figure 1.**

<table>
<thead>
<tr>
<th>C</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>1</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>000</td>
</tr>
<tr>
<td>3</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>4</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>5</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>6</td>
<td>000</td>
<td>000</td>
</tr>
</tbody>
</table>

Codes:

- $\uparrow$ 000 0 or 0 0 0 0 1 0 1 0 1 0 1 0 1
- $\downarrow$ 0 0 1 0 0 1 0
- $\leftarrow$ 0 1 1 1 1 1
- $\rightarrow$ 1 0 0
- $\leftarrow$ 1 0 1
- $\rightarrow$ 1 1 0
- $\leftarrow$ 1 1 1

This vector cannot be a plot vector or a move up (↑).

**Figure 2.**

<table>
<thead>
<tr>
<th>C</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0011</td>
<td>1111</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>0000</td>
</tr>
<tr>
<td>3</td>
<td>0110</td>
<td>0100</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>0110</td>
</tr>
<tr>
<td>5</td>
<td>0000</td>
<td>0010</td>
</tr>
<tr>
<td>6</td>
<td>0000</td>
<td>0010</td>
</tr>
<tr>
<td>7</td>
<td>0000</td>
<td>0010</td>
</tr>
<tr>
<td>8</td>
<td>0000</td>
<td>0010</td>
</tr>
<tr>
<td>9</td>
<td>0000</td>
<td>0010</td>
</tr>
</tbody>
</table>

Hexadecimal Codes:

- 0000 → 0
- 0001 → 1
- 0010 → 2
- 0011 → 3
- 0100 → 4
- 0101 → 5
- 0110 → 6
- 0111 → 7
- 1000 → 8
- 1001 → 9
- 1010 → A
- 1011 → B
- 1100 → C
- 1101 → D
- 1110 → E
- 1111 → F

---

53
ROD'S COLOR PATTERN

PROGRAM DESCRIPTION

ROD'S COLOR PATTERN is a simple but eloquent program. It generates a continuous flow of colored mosaic-like patterns in a 40 high by 40 wide block matrix. Many of the patterns generated by this program are pleasing to the eye and will dazzle the mind for minutes at a time.

REQUIREMENTS

4K or greater Apple II system with a color video display.
BASIC is the programming language used.

PROGRAM LISTING

100 40
105 FOR K=1 TO 40
110 FOR I=1 TO 40
115 FOR J=1 TO 40
120 #=1+1
125 COLOR=#*(12) MOD 12
130 PLOT I,K; PLOT K,I; PLOT 40 K -1,40 -K
135 PLOT 40 -K,40 -I; PLOT K,40 -I;
140 NEXT J
145 NEXT I
150 NEXT K: GOTO 105
PROGRAM LISTING: PONG

5 ACH PONG BY WENDELL BITTER
10 REM    7/2/77
15 REM  PARALLEL SWITCHES CONTROL
          PARALLEL SIZE AFTER A MISS
          OR DURING A HIT
20 GR
25 DIM P(3) D1N H(5)(58)
30 A=0:G05=1:1C=1
35 COLOR=0:HLIN 1.58 AT 3: HLIN
1.36 AT 31
40 CALL _VOR 12 prar 83: INPUT "HANDLE"
          ALL OR PONG? 1, A,P(7)
45 INPUT "PARALLEL SIZE (1-6) ? ", P(7)
PS: IF PS(1) OR PS(2) THEN 45
    1:PS(1)=1
50 CALL _VOR 12
55 IF A=3 THEN 285
60 H(1)=COLOR=125:HLIN 0.39 AT
20:GOTO 260
65 FEEC H(4) TO 0 STEP 6
70 Y=Y+Y: IF Y1 AND Y2 THEN
60: IF Y<1 THEN Y=1: IF Y>288
    THEN Y=288
75 V=V+1 FOR T=1 TO 75: PRINT "PERT"
    X=163000: NEXT T
80 IF A=4 OR X=60 THEN 210:COLOR=
    6:PL0T X,Y:VY=COLOR=150:PL0T
X,Y
85 Y=V: IF X<10 THEN 490: GO TO
89: NEXT A
90 K0SUB 288
95 IF SC0RM(N,Y<1)+0 AND Y=1
    THEN 165
100 FOR T=1 TO 18:PRINT X=163000
    : NEXT T
105 IF H AND G THEN 136
110 IF P=32 THEN 0: IF X=PP+1
        THEN V=2: IF V=PP+2 THEN V=
1
120 IF Y=PP+3 THEN V=1: IF V=PP+
4 THEN V=2: IF V=PP+5 THEN
    V=3
125 IF S=0 THEN V=105: END (7)
130 COLOR=6: PL0T X=0:Y
135 IF (K AND D2) OR (Y<15) AND V=0
    THEN V=4: END (9)
140 IF A=6 THEN V=V=ABS (W)
145 R=90:PL0T X=3A:Y=R=0
150 IF PEEX (-16288) AND S=1
    THEN S=2
155 IF PEEX (-16287) AND S=1
    THEN S=5
160 GOTO 65
165 COLOR=1: PL0T X=0:Y
170 COLOR=55: PL0T X,Y+Y
1 AND Y+Y (180)
175 FOR T=1 TO 75: PRINT (-16334)
>PEEX (-16333): PEEX (-16336)
V: NEXT T
180 IF V=0 THEN SR=9:IF V=30
    THEN SL=9:0
185 Y=23: TRS 7: PRINT SL="": THE
70 PRINT SR
190 COLOR=6: PL0T X=0:Y
195 IF P=15 OR S=12 THEN 260
200 COLOR=1: PL0T X,Y+Y:PP Y=1
    AND Y+Y (180)
205 FOR T=1 TO 75: IF T MOD 540
    THEN 210: IF PEEX (-16336)
7127 AND S=5 THEN 225:IF
    PEEX (-16281) AND S=5
    THEN S=1
210 GOSUB 235: NEXT T
215 V=V+1: IF V=0 THEN 9=V1:1

220 IF M THEN V1= RND (137)
225 V=1: RND (2)
230 GOTO 65

56
COLOR SKETCH

PROGRAM DESCRIPTION
Color Sketch is a little program that transforms the Apple II into an artist's easel, the screen into a sketch pad. The user as an artist has a 40 high by 40 wide (1600 blocks) sketching pad to fill with a rainbow of fifteen colors. Placement of colors is determined by controlling paddle inputs; one for the horizontal and the other for the vertical. Colors are selected by depressing a letter from A through P on the keyboard.

An enormous number of distinct pictures can be drawn on the sketch pad and this program will provide many hours of visual entertainment.

REQUIREMENTS
This program will fit into a 4K system in the BASIC mode.
15 DIM BK(40): TEXT : CALL -336
20 CALL -96: GOTO 90
29 CALL -336: GOTO 90
49 BE="***************
***************: RETURN
50 BE="COLOR SKETCH": RETURN
59 BE="COPYRIGHT APPLE COMPUTER 1977": RETURN
40 BE="THIS PROGRAM ALLOWS YOU TO " : RETURN
96 BA="SKETCH COLORED FIGURES IN"
55 BA="LOW RESOLUTION GRAPHICS WITH PADDLES": RETURN
60 AX=261:TON=58: GOSUB 85: RETURN
65 AX=10:TON=10: GOSUB 05: RETURN
70 AX=20:TON=50: GOSUB 05:*AX=30 :TON=0: GOSUB 85: RETURN
75 AX=20:TON=50: GOSUB 05: RETURN
80 AX=0:TON=258: GOSUB 05:*AX=9
85 AX=258: GOSUB 05: RETURN
95 POKE 1,TON MOD 256: POKE 24 ,TON=256+1: POKE 8,11: CALL 3: RETURN
90 GOSUB 38: GOSUB 25: PRINT :
...
MASTERMIND PROGRAM

PROGRAM DESCRIPTION
MASTERMIND is a game of strategy that matches your wits against Apple's. The object of the game is to choose correctly which 5 colored bars have been secretly chosen by the computer. Eight different colors are possible for each bar - Red (R), Yellow (Y), Violet (V), Orange (O), White (W), and Black (B). A color may be used more than once. Guesses for a turn are made by selecting a color for each of the five hidden bars. After hitting the RETURN key Apple will indicate the correctness of the turn. Each white square to the right of your turn indicates a correctly colored and positioned bar. Each grey square acknowledges a correctly colored but improperly positioned bar. No squares indicate you're way off.

Test your skill and challenge the Apple II to a game of MASTERMIND.

REQUIREMENTS
8K or greater Apple II computer system.
BASIC is the programming language.
A REN GAME OF MASTERMIND 8-85-77

POKE (APPLE COMPUTER)

10 DIM X(8),Y(8),A[8,8],B[8],C[8]
20 X(I)=1:Y(I)=1:Z(I)=1:W(I)=1
30 X(I)=1:Y(I)=1:Z(I)=1:W(I)=1
40 X(I)=1:Y(I)=1:Z(I)=1:W(I)=1
50 X(I)=1:Y(I)=1:Z(I)=1:W(I)=1

60 REM CALL -804: PRINT " ;

70 PRINT "WELCOME TO THE GAME OF MASTERMIND!

80 PRINT "YOU MUST GUESS 5 COLOR
90 PRINT "SHAPES"

100 PRINT "I WILL GIVE YOU THE FIRST
110 PRINT "NUMBER OF GUESSES. THEN I
120 PRINT "WILL GIVE YOU EIGHT DIFFERENT COLORS
130 PRINT "TO GUESS FROM."

140 PRINT "";

150 PRINT "FEWER THAN 7 GUESSES--EXC
160 PRINT "ELENT"; PRINT " 7 TO 9 GUESSE
170 PRINT "S---GOOD"; PRINT " 10 TO 14 GUE
180 PRINT "SES---AVERAGE";

190 PRINT "MORE THAN 14 GUESSES--POOR";

200 CALL -396; TAB 7; PRINT

"HIT ANY KEY TO BEGIN PLAY"

210 CALL -396: IF PEEK (-16004)
220 THEN 100: POKE -16388,
230 OR : PRINT I FOR I=1 TO
240: PRINT X(I)+Y(I)+Z(I)+W(I)
250: PRINT "X";X(I);Y(I);Z(I)
260: PRINT ";";X(I);Y(I);Z(I);W(I)

270 PRINT "TRY 1: PRINT ";

280 TRY=1: TRY+1: PRINT " ";

290 CALL 396: TAB 5: PRINT " ";

300 CALL 396: TAB 5: PRINT " ";

310 REM CALL -304 SETS INVERSE VID
320 REM CALL -308 SETS NORMAL VID
330 REM PEEK (-15384) IS KEY (ASCII)!
340 REM IF I > 1 THEN STROBE SET
350 REM POKE-16388 CLR'S AND STROBE
360 REM CALL-996 CLEAR'S SCREEN AND
370 REM CURSOR TO UPPER LEFT.
380 REM IN 310, KEY=6-29= -1 OR +1
390 REM CALL KEY=198 OR 149 ASCII
400 REM STATS 19-39 INTRD
410 REM STATS 100-110 NEW SETUP
420 REM STATS 1266 NEW GUESS
430 REM STATS 1266-1288 GUESH INPUT
440 REM STAT 396 GUESS CVR.
450 REM STATS 300-314 NEW CVR.
460 REM SBUX 1600 COLOR LINE
470 REM SBUX 2000 MATCH TEST

480 REM CALL -304 SETS INVERSE VID
490 REM CALL -308 SETS NORMAL VID
500 REM PEEK (-15384) IS KEY (ASCII)!
510 REM IF I > 1 THEN STROBE SET
520 REM POKE-16388 CLR'S AND STROBE
530 REM CALL-996 CLEAR'S SCREEN AND
540 REM CURSOR TO UPPER LEFT.
550 REM IN 310, KEY=6-29= -1 OR +1
560 REM CALL KEY=198 OR 149 ASCII
570 REM STATS 19-39 INTRD
580 REM STATS 100-110 NEW SETUP
590 REM STATS 1266 NEW GUESS
600 REM STATS 1266-1288 GUESH INPUT
610 REM STAT 396 GUESS CVR.
620 REM STATS 300-314 NEW CVR.
630 REM SBUX 1600 COLOR LINE
640 REM SBUX 2000 MATCH TEST

650 REM CALL -304 SETS INVERSE VID
660 REM CALL -308 SETS NORMAL VID
670 REM PEEK (-15384) IS KEY (ASCII)!
680 REM IF I > 1 THEN STROBE SET
690 REM POKE-16388 CLR'S AND STROBE
700 REM CALL-996 CLEAR'S SCREEN AND
710 REM CURSOR TO UPPER LEFT.
720 REM IN 310, KEY=6-29= -1 OR +1
730 REM CALL KEY=198 OR 149 ASCII
740 REM STATS 19-39 INTRD
750 REM STATS 100-110 NEW SETUP
760 REM STATS 1266 NEW GUESS
770 REM STATS 1266-1288 GUESH INPUT
780 REM STAT 396 GUESS CVR.
790 REM STATS 300-314 NEW CVR.
800 REM SBUX 1600 COLOR LINE
810 REM SBUX 2000 MATCH TEST

820 REM CALL -304 SETS INVERSE VID
830 REM CALL -308 SETS NORMAL VID
840 REM PEEK (-15384) IS KEY (ASCII)!
850 REM IF I > 1 THEN STROBE SET
860 REM POKE-16388 CLR'S AND STROBE
870 REM CALL-996 CLEAR'S SCREEN AND
880 REM CURSOR TO UPPER LEFT.
890 REM IN 310, KEY=6-29= -1 OR +1
900 REM CALL KEY=198 OR 149 ASCII
910 REM STATS 19-39 INTRD
920 REM STATS 100-110 NEW SETUP
930 REM STATS 1266 NEW GUESS
940 REM STATS 1266-1288 GUESH INPUT
950 REM STAT 396 GUESS CVR.
960 REM STATS 300-314 NEW CVR.
970 REM SBUX 1600 COLOR LINE
980 REM SBUX 2000 MATCH TEST

990 REM CALL -304 SETS INVERSE VID
1000 REM CALL -308 SETS NORMAL VID
1010 REM PEEK (-15384) IS KEY (ASCII)!
1020 REM IF I > 1 THEN STROBE SET
1030 REM POKE-16388 CLR'S AND STROBE
1040 REM CALL-996 CLEAR'S SCREEN AND
1050 REM CURSOR TO UPPER LEFT.
1060 REM IN 310, KEY=6-29= -1 OR +1
1070 REM CALL KEY=198 OR 149 ASCII
1080 REM STATS 19-39 INTRD
1090 REM STATS 100-110 NEW SETUP
1100 REM STATS 1266 NEW GUESS
1110 REM STATS 1266-1288 GUESH INPUT
1120 REM STAT 396 GUESS CVR.
1130 REM STATS 300-314 NEW CVR.
1140 REM SBUX 1600 COLOR LINE
1150 REM SBUX 2000 MATCH TEST

1160 REM CALL -304 SETS INVERSE VID
1170 REM CALL -308 SETS NORMAL VID
1180 REM PEEK (-15384) IS KEY (ASCII)!
1190 REM IF I > 1 THEN STROBE SET
1200 REM POKE-16388 CLR'S AND STROBE
1210 REM CALL-996 CLEAR'S SCREEN AND
1220 REM CURSOR TO UPPER LEFT.
1230 REM IN 310, KEY=6-29= -1 OR +1
1240 REM CALL KEY=198 OR 149 ASCII
1250 REM STATS 19-39 INTRD
1260 REM STATS 100-110 NEW SETUP
1270 REM STATS 1266 NEW GUESS
1280 REM STATS 1266-1288 GUESH INPUT
1290 REM STAT 396 GUESS CVR.
1300 REM STATS 300-314 NEW CVR.
1310 REM SBUX 1600 COLOR LINE
1320 REM SBUX 2000 MATCH TEST
PROGRAM DESCRIPTION
This program plots three Biorhythm functions: Physical (P), Emotional (E), and Mental (M) or intellectual. All three functions are plotted in the color graphics display mode.

Biorhythm theory states that aspects of the mind run in cycles. A brief description of the three cycles follows:

Physical
The Physical Biorhythm takes 23 days to complete and is an indirect indicator of the physical state of the individual. It covers physical well-being, basic bodily functions, strength, coordination, and resistance to disease.

Emotional
The Emotional Biorhythm takes 28 days to complete. It indirectly indicates the level of sensitivity, mental health, mood, and creativity.

Mental
The mental cycle takes 33 days to complete and indirectly indicates the level of alertness, logic and analytic functions of the individual, and mental receptivity.

Biorhythms
Biorhythms are thought to affect behavior. When they cross a "baseline" the functions change phase - become unstable - and this causes Critical Days. These days are, according to the theory, our weakest and most vulnerable times. Accidents, catching colds, and bodily harm may occur on physically critical days. Depression, quarrels, and frustration are most likely on emotionally critical days. Finally, slowness of the mind, resistance to new situations and unclear thinking are likely on mentally critical days.

REQUIREMENTS
This program fits into a 4K or greater system.
BASIC is the programming language used.
PROGRAM LISTING: BIORHYTHM

5 POKE 2,173; POKE 3,68; POKE 4,193; POKE 5,165; POKE 6,8
7 POKE 7,32; POKE 8,150; POKE 9,256; POKE 10,168; POKE 11,14
14 POKE 12,200; POKE 13,6
16 POKE 14,199; POKE 15,24; POKE 16,249; POKE 17,5; POKE 18,1
20 POKE 19,11; POKE 20,75; POKE 21,2; POKE 22,6; POKE 23,25
25 GOTO 25
28 TI=31: GOSUB 30: RETURN
35 PRINT "***************
***************": RETURN
39 XX=0: TIOM=000: GOSUB 40: RETURN
43 XX=0: TIOM=000: GOSUB 40: RETURN
48 POKE 1,781: POKE 255: POKE 21,4: POKE 3,4: CALL 21: RETURN
53 A=19: C=10: B=100: X=1000
57 D=11: C=120: B=1000: X=1000
61 =X100
65 A=X+80: B=X+10: RETURN
69 XX=0: TM=15: GOSUB 70: RETURN
73 XX=7: TM=10: GOSUB 70: RETURN
76 POKE 1,71: MOV 256: POKE 24,
78 T=L2+1: POKE 0,KK: CALL 2
79 RETURN
83 GOSUB 60: INPUT "DATE (M,D,Y) ":
87 M,D,Y= TI=TIM: I=I+1
86 H=V=0: X=0: Y=0: SMMY=I+1
87 =X5+1: N=2: IF M<8 THEN
89 N=0: N=0: RETURN
95 S M=M+10: SMMY=SMMY: SMMY=SMMY+10
100 SMMY=SMMY: TEXT 1: CALL 1061
101 POKE 94,261: GOSUB 26: GOSUB 25: RETURN
106 PRINT 1: PAR II BIORHYTHM (C4)
111 ": TAB 15: PRINT
115 GOSUB 25: TAB 5: PRINT "COPYRIGHT
119 1977 APPLE COMPUTER INC.":
123 POKE 34,24: VTAB 24
127 GOSUB 60: INPUT "NAME ":,A,
131 : VTAB 22: PRINT H: VTAB 24
135 PRINT "EINSTEIN": GOSUB 75
140 : VTAB 22: TAB 11: PRINT "BIRTH 
144 D A T E " :Y,"",Y,":",Y:VTAB
148 H=H+H: CALL -658
152 VTAB "FORECAST ":; GOSUB 75
156 A=N+88: IF B<8 THEN N=N+2532
159 VTAB 23: TAB 15: PRINT "FORECAST
163 ST DATE ":X,"",Y,"",Y:VTAB
167 TO CALL -658
171
PROGRAM DESCRIPTION
DRAGON MAZE is a game that will test your skill and memory. A maze is constructed on the video screen. You watch carefully as it is completed. After it is finished the maze is hidden as if the lights were turned out. The object of the game is to get out of the maze before the dragon eats you. A reddish-brown square indicates your position and a purple square represents the dragon's.* You move by hitting a letter on the keyboard; U for up, D for down, R for right, and L for left. As you advance so does the dragon. The scent of humans drives the dragon crazy; when he is enraged he breaks through walls to get at you. DRAGON MAZE is not a game for the weak at heart. Try it if you dare to attempt out-smarting the dragon.

REQUIREMENTS
8K or greater Apple II computer system.
BASIC is the programming language.

* Color tints may vary depending upon video monitor or television adjustments.
PROGRAM LISTING: DRAGON MAZE

1  TEXT : CALL -906
2  PRINT "WELCOME TO THE DRAGON'S MAZE!"
3  PRINT "YOU MAY WATCH WHILE I BUILD A MAZE."
4  PRINT "BUT WHEN IT'S COMPLETE, I'LL ERASE."
5  PRINT "THE PICTURE. THEN YOU'LL ONLY SEE THE WALLS AS YOU BUMP INTO THEM."
6  PRINT "TO MOVE, YOU HIT 'A' FOR RIGHT, 'L' FOR LEFT, 'U' FOR UP, AND 'D' FOR DOWN. DO NOT HIT RETURN."
7  PRINT "THE OBJECT IS FOR YOU (THE GREEN DOT)."
8  PRINT "THE DOOR ON THE RIGHT SIDE"
9  PRINT "BEFORE THE DRAGON (THE RED DOT) EATS!
10  PRINT "YOU."
11  PRINT "BEWARE!!!!!!! SOMETIMES THE DRAGON EATS REAL RABBIT."
12  PRINT "GETS REAL MAD, AND CLIMBS OVER A WALL."
13  PRINT "BUT MOST OF THE TIME, HE CAN'T DO OVER."
14  PRINT "YOU HAVE TO GO AROUND."
15  PRINT "YOU CAN ODDLY TELL WHERE A WALL."

20  PRINT "IS, EVEN BEFORE YOU CAN SEE IT, BY"
21  PRINT "THE FACT THAT THE DRAGON CAN'T GET"
22  PRINT "THROUGH IT."
23  PRINT "GOTO 12,940: PRINT "GAME OVER"
24  PRINT "GET OUT OF HERE!"
25  PRINT "ANSWERS:
26  PRINT "MAZE IS 3X9 AT 3x3 AT 3X3."
27  PRINT "PLAY AGAIN?"
28  PRINT "MAZE Of 3X9 AT 3X3 AT 3X3."
DRAGON MAZE cont.

1225 XR=3*X-2:HY=3*Y-2
1230 BY=RND (13)+1
1240 COLOR=6:NLIN 3*XW-2,3*YW-1
AT 33
1250 SX=13:SY=W
1260 DX=3*DX-2:DY=3*SY-2
1270 RX=1
1280 K: PEKB (-16384): IF K<128 THEN
1500 1510 POKE -16386,8
1515 DX=K: GOSUB 7000:K=0
1520 IF SX+R AND SY=E THEN 3300
1525 IF K: ASC("r") THEN 2000
1530 IF K: ASC("p") THEN 2900
1535 IF K: ASC("L") THEN 2500
1540 IF K: ASC("u") THEN 3000
1550 IF K: ASC("b") THEN 3500
1555 GOSUB 3988:GOTO 1500
2000 DX=1:DY=0
2010 IF (MX+13*AM-1) AND 14 THEN
4000
2020 FX=3*X-2:FY=3*Y-2: FOR I=1 TO
3
2030 FX=FX+DX:FY=FY+DY
2340 COLOR=0
2360 FOR X=0 TO I: FOR Y=0 TO I:
2361 POKE X+AM:HYL: NEXT L,K: COLOR=
2: FOR K>0 TO I: FOR L>0 TO I:
2363 FOR K=0 TO 1: FOR L=0 TO I:
2365 IF HY=FY THEN
2367 NEXT I
2370 NEXT 1
2375 X=3*X+Y+DY
2380 IF X=13 AND Y=0 THEN 3000
2385 GOTO 1500
2390 DX=-1:DY=0
2510 IF (MX+13*AM-1) AND 14 THEN
4100
2520 GOTO 2020
3000 DX=0:DY=-1
3010 IF MX+13*AM(Y-2))<10 THEN 4200
3020 GOTO 2020
3500 DX=0:DY=1
3510 IF MX+13*AM(Y-1)<10 THEN 4300
7000 IF X<13 THEN 7005: IF Y<SY THEN
7080
7090 IF Y<SY THEN 7100: IF Y<SY THEN
7150
7400 IF SX<13 THEN 7800: IF TCSX=
1345SY-1)>SY THEN 7810: IF
7840 MCSX+13*SY-1)MOD 10 THEN
7850
7860 DX=1:DY=0
7870 COLOR=8
7880 POKE 3*SY-2:RY=3*SY-2
7890 FOR I=1 TO 3:POKE3*X:RX:RY=RY+9
7924 COLOR=0
7925 FOR K=8 TO 1: FOR L=0 TO 1:
7930 POKE 3*X,YL: NEXT L,K: COLOR=
80: FOR K>8 TO 1: FOR L=0 TO
1: POKE 3*X,YL: NEXT L,K:
8187:ORY=RY
7928 NEXT I
7935 SX=5*X:SY=SY+8
7950 TSX=13*SY-1)=TSX=13*SY-1)+1
7965 RETURN
7950 IF SY<13 THEN 7100: IF TCSX=
13*SY-1)>SY THEN 7860: IF
7980 MCSX+13*SY-1)MOD 10 THEN 7100
7990 DX=6:SY=1:GOTO 7850
7180 IF SX<1 THEN 7150: IF TCSX=
13*SY-1)>SY THEN 7110: IF
7190 MCSX+13*SY-1)MOD 10 THEN
7150
7200 PRINT "YOU WIN!"
7210 PRINT "SCORE=":S+3
7220 END

DRAGON MAZE cont.

7118 ox=-1:dy=-1:goto 7620
7130 if sy=1 then 7085: if tsx+13*(sy-1)+9 then 7160: if
7135 (sx+13*(sy-1)-13)/10 then
7140
7160 ox=ox:dy=dy:goto 7620
8000 gosub 5000: gosub 5000: gosub
8005: gosub 5000: print "the bee
got you!"
8099 end
APPLE II FIRMWARE

1. System Monitor Commands
2. Control and Editing Characters
3. Special Controls and Features
4. Annotated Monitor and Dis-assembler Listing
5. Binary Floating Point Package
6. Sweet 16 Interpreter Listing
7. 6502 Op Codes
System Monitor Commands

Apple II contains a powerful machine level monitor for use by the advanced programmer. To enter the monitor either press RESET button on keyboard or CALL-151 (Hex FF65) from Basic. Apple II will respond with an "*" (asterisk) prompt character on the TV display. This action will not kill current BASIC program which may be re-entered by a C (control C). NOTE: "adrs" is a four digit hexadecimal number and "data" is a two digit hexadecimal number. Remember to press "return" button at the end of each line.

<table>
<thead>
<tr>
<th>Command Format</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examine Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrs</td>
<td>*C0F2</td>
<td>Examines (displays) single memory location of (adrs)</td>
</tr>
<tr>
<td>adrs1.adrs2</td>
<td>*1024.1048</td>
<td>Examines (displays) range of memory from (adrs1) thru (adrs2)</td>
</tr>
<tr>
<td>(return)</td>
<td>*(return)</td>
<td>Examines (displays) next 8 memory locations.</td>
</tr>
<tr>
<td>.adrs2</td>
<td>*.4096</td>
<td>Examines (displays) memory from current location through location (adrs2)</td>
</tr>
<tr>
<td>Change Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrs:data</td>
<td>*A256:EF 20 43</td>
<td>Deposits data into memory starting at location (adrs).</td>
</tr>
<tr>
<td>data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:data data</td>
<td>*:F0 A2 12</td>
<td>Deposits data into memory starting after (adrs) last used for deposits.</td>
</tr>
<tr>
<td>Move Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrs1&lt;adrs2.</td>
<td>*100&lt;B010.B410M</td>
<td>Copy the data now in the memory range from (adrs2) to (adrs3) into memory</td>
</tr>
<tr>
<td>adrs3M</td>
<td></td>
<td>locations starting at (adrs1).</td>
</tr>
<tr>
<td>Verify Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrs1&lt;adrs2</td>
<td>*100&lt;B010.B410V</td>
<td>Verify that block of data in memory range from (adrs2) to (adrs3) exactly</td>
</tr>
<tr>
<td>adrs3V</td>
<td></td>
<td>matches data block starting at memory location (adrs1) and displays</td>
</tr>
<tr>
<td></td>
<td></td>
<td>differences if any.</td>
</tr>
<tr>
<td>Command Format</td>
<td>Example</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Cassette I/O</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrsl.adrs2R</td>
<td>*300.4FFR</td>
<td>Reads cassette data into specified memory (adrs) range. Record length must be same as memory range or an error will occur.</td>
</tr>
<tr>
<td>adrsl.adrs2W</td>
<td>*800.9FFW</td>
<td>Writes onto cassette data from specified memory (adrs) range.</td>
</tr>
<tr>
<td><strong>Display</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>*I</td>
<td>Set inverse video mode. (Black characters on white background)</td>
</tr>
<tr>
<td>M</td>
<td>*N</td>
<td>Set normal video mode. (White characters on black background)</td>
</tr>
<tr>
<td><strong>Dis-assembler</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrsl</td>
<td>*C800L</td>
<td>Decodes 20 instructions starting at memory (adrs) into 6502 assembly mnemonic code.</td>
</tr>
<tr>
<td>L</td>
<td>*L</td>
<td>Decodes next 20 instructions starting at current memory address.</td>
</tr>
<tr>
<td><strong>Mini-assembler</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Turn-on)</td>
<td>*F666G</td>
<td>Turns-on mini-assembler. Prompt character is now a &quot;!&quot; (exclamation point).</td>
</tr>
<tr>
<td>$(monitor:</td>
<td>$C800L</td>
<td>Executes any monitor command from mini-assembler then returns control to mini-assembler. Note that many monitor commands change current memory address reference so that it is good practice to retype desired address reference upon return to mini-assembler.</td>
</tr>
<tr>
<td>command)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addr:(6502</td>
<td>!C010:STA 23FF</td>
<td>Assembles a mnemonic 6502 instruction into machine codes. If error, machine will refuse instruction, sound bell, and reprint line with up arrow under error.</td>
</tr>
<tr>
<td>MNEMONIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>instruction)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command Format</td>
<td>Example</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>(space) (65Ø2 mnemonic instruction)</td>
<td>! STA ØlFF</td>
<td>Assembles instruction into next available memory location. (Note space between &quot;f&quot; and instruction)</td>
</tr>
<tr>
<td>(TURN-OFF)</td>
<td>! (Reset Button)</td>
<td>Exits mini-assembler and returns to system monitor.</td>
</tr>
</tbody>
</table>

**Monitor Program Execution and Debugging**

<table>
<thead>
<tr>
<th>adrSG</th>
<th>*30ØG</th>
<th>Runs machine level program starting at memory (adrs).</th>
</tr>
</thead>
<tbody>
<tr>
<td>adrST</td>
<td>*8ØØT</td>
<td>Traces a program starting at memory location (adrs) and continues trace until hitting a breakpoint. Break occurs on instruction ØØ (BRK), and returns control to system monitor. Opens 65Ø2 status registers (see note 1)</td>
</tr>
<tr>
<td>asrdS</td>
<td>*CØ5ØS</td>
<td>Single steps through program beginning at memory location (adrs). Type a letter S for each additional step that you want displayed. Opens 65Ø2 status registers (see Note 1).</td>
</tr>
<tr>
<td>(Control E)</td>
<td>*EC</td>
<td>Displays 65Ø2 status registers and opens them for modification (see Note 1)</td>
</tr>
<tr>
<td>(Control Y)</td>
<td>*YC</td>
<td>Executes user specified machine language subroutine starting at memory location (3F8).</td>
</tr>
</tbody>
</table>

**Note 1:**

65Ø2 status registers are open if they are last line displayed on screen. To change them type ":" then "data" for each register.

Example:  
A = 3C  X = FF  Y = ØØ  P = 32  S = F2  
*: FF  Changes A register only  
*: FF ØØ 33  Changes A, X, and Y registers

To change S register, you must first retype data for A, X, Y and P.

**Hexidecimal Arithmetic**

<table>
<thead>
<tr>
<th>datal+data2</th>
<th>*78+34</th>
<th>Performs hexidecimal sum of datal plus data2.</th>
</tr>
</thead>
<tbody>
<tr>
<td>datal-data2</td>
<td>*AE-34</td>
<td>Performs hexidecimal difference of datal minus data2.</td>
</tr>
<tr>
<td>Command Format</td>
<td>Example</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Set Input/Output Ports</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(X) (Control P)</td>
<td>*5PC</td>
<td>Sets printer output to I/O slot number (X). (see Note 2 below)</td>
</tr>
<tr>
<td>(X) (Control K)</td>
<td>*2KC</td>
<td>Sets keyboard input to I/O slot number (X). (see Note 2 below)</td>
</tr>
</tbody>
</table>

**Note 2:**

Only slots 1 through 7 are addressable in this mode. Address Ø (Ex: ØPC or ØKC) resets ports to internal video display and keyboard. These commands will not work unless Apple II interfaces are plugged into specified I/O slot.

**Multiple Commands**

- *100L 400G AFFT
  - Multiple monitor commands may be given on same line if separated by a "space".
- *LLLLL
  - Single letter commands may be repeated without spaces.
SPECIAL CONTROL AND EDITING CHARACTERS

"Control" characters are indicated by a super-scripted "C" such as G\(^C\). They are obtained by holding down the CTRL key while typing the specified letter. Control characters are NOT displayed on the TV screen. B\(^C\) and C\(^C\) must be followed by a carriage return. Screen editing characters are indicated by a sub-scripted "E" such as D\(_E\). They are obtained by pressing and releasing the ESC key then typing specified letter. Edit characters send information only to display screen and does not send data to memory. For example, U\(_E\) moves to cursor to right and copies text while A\(_E\) moves cursor to right but does not copy text.

<table>
<thead>
<tr>
<th>CHARACTER</th>
<th>DESCRIPTION OF ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET key</td>
<td>Immediately interrupts any program execution and resets computer. Also sets all text mode with scrolling window at maximum. Control is transferred to System Monitor and Apple prompts with a &quot;*&quot; (asterisk) and a bell. Hitting RESET key does NOT destroy existing BASIC or machine language program.</td>
</tr>
<tr>
<td>Control B</td>
<td>If in System Monitor (as indicated by a &quot;*&quot;), a control B and a carriage return will transfer control to BASIC, scratching (killing) any existing BASIC program and set HIMEM: to maximum installed user memory and LOMEM: to 2048.</td>
</tr>
<tr>
<td>Control C</td>
<td>If in BASIC, halts program and displays line number where stop occurred*. Program may be continued with a CON command. If in System Monitor, (as indicated by &quot;*&quot;), control C and a carriage return will enter BASIC without killing current program.</td>
</tr>
<tr>
<td>Control G</td>
<td>Sounds bell (beeps speaker)</td>
</tr>
<tr>
<td>Control H</td>
<td>Backspaces cursor and deletes any overwritten characters from computer but not from screen. Apply supplied keyboards have special key &quot;4-.&quot; on right side of keyboard that provides this function without using control button.</td>
</tr>
<tr>
<td>Control J</td>
<td>Issues line feed only</td>
</tr>
<tr>
<td>Control V</td>
<td>Compliment to H(^C). Forward spaces cursor and copies over written characters. Apple keyboards have &quot;+&quot; key on right side which also performs this function.</td>
</tr>
<tr>
<td>Control X</td>
<td>Immediately deletes current line.</td>
</tr>
</tbody>
</table>

* If BASIC program is expecting keyboard input, you will have to hit carriage return key after typing control C.
<table>
<thead>
<tr>
<th>CHARACTER</th>
<th>DESCRIPTION OF ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE</td>
<td>Move cursor to right</td>
</tr>
<tr>
<td>BE</td>
<td>Move cursor to left</td>
</tr>
<tr>
<td>CE</td>
<td>Move cursor down</td>
</tr>
<tr>
<td>DE</td>
<td>Move cursor up</td>
</tr>
<tr>
<td>EE</td>
<td>Clear text from cursor to end of line</td>
</tr>
<tr>
<td>FE</td>
<td>Clear text from cursor to end of page</td>
</tr>
<tr>
<td>@E</td>
<td>Home cursor to top of page, clear text to end of page</td>
</tr>
</tbody>
</table>
## Special Controls and Features

<table>
<thead>
<tr>
<th>Hex</th>
<th>BASIC Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C050</td>
<td>10  POKE -16304,0</td>
<td>Set color graphics mode</td>
</tr>
<tr>
<td>C051</td>
<td>20  POKE -16303,0</td>
<td>Set text mode</td>
</tr>
<tr>
<td>C052</td>
<td>30  POKE -16302,0</td>
<td>Clear mixed graphics</td>
</tr>
<tr>
<td>C053</td>
<td>40  POKE -16301,0</td>
<td>Set mixed graphics (4 lines text)</td>
</tr>
<tr>
<td>C054</td>
<td>50  POKE -16300,0</td>
<td>Clear display Page 2 (BASIC commands use Page 1 only)</td>
</tr>
<tr>
<td>C055</td>
<td>60  POKE -16299,0</td>
<td>Set display to Page 2 (alternate)</td>
</tr>
<tr>
<td>C056</td>
<td>70  POKE -16298,0</td>
<td>Clear HIRES graphics mode</td>
</tr>
<tr>
<td>C057</td>
<td>80  POKE -16297,0</td>
<td>Set HIRES graphics mode</td>
</tr>
</tbody>
</table>

### TEXT Mode Controls

<p>| 0020 | 90  POKE 32,L1                     | Set left side of scrolling window to location specified by L1 in range of 0 to 39. |
| 0021 | 100 POKE 33,W1                     | Set window width to amount specified by W1. L1+W1&lt;40. W1&gt;0 |
| 0022 | 110 POKE 34,11                     | Set window top to line specified by T1 in range of 0 to 23 |
| 0023 | 120 POKE 35,B1                     | Set window bottom to line specified by B1 in the range of 0 to 23. B1&gt;T1 |
| 0024 | 130 CH=PEEK(36)                    | Read/set cursor horizontal position in the range of 0 to 39. If using TAB, you must add &quot;1&quot; to cursor position read value; Ex. 140 and 150 perform identical function. |
| 140  | POKE 36,CH                          | |
| 150  | TAB(CH+1)                           | |
| 0025 | 160 CV=PEEK(37)                     | Similar to above. Read/set cursor vertical position in the range 0 to 23. |
| 170  | POKE 37,CV                          | |
| 180  | VTAB(CV+1)                          | |
| 0032 | 190 POKE 50,127                     | Set inverse flag if 127 (Ex. 190) |
| 200  | POKE 50,255                         | Set normal flag if 255(Ex. 200) |
| FC58 | 210 CALL -936                       | (@E) Home cursor, clear screen |
| FC42 | 220 CALL -958                       | (FE) Clear from cursor to end of page |</p>
<table>
<thead>
<tr>
<th>Hex</th>
<th>BASIC Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC9C</td>
<td>230 CALL -868</td>
<td>(EF) Clear from cursor to end of line</td>
</tr>
<tr>
<td>FC66</td>
<td>240 CALL -922</td>
<td>(JC) Line feed</td>
</tr>
<tr>
<td>FC70</td>
<td>250 CALL -912</td>
<td>Scroll up text one line</td>
</tr>
</tbody>
</table>

**Miscellaneous**

<table>
<thead>
<tr>
<th>Hex</th>
<th>BASIC Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C030</td>
<td>360 X=PEEK(-16336)</td>
<td>Toggle speaker</td>
</tr>
<tr>
<td></td>
<td>365 POKE -16336,0</td>
<td></td>
</tr>
<tr>
<td>C000</td>
<td>370 X=PEEK(-16384)</td>
<td>Read keyboard; if X&gt;127 then key was pressed.</td>
</tr>
<tr>
<td>C010</td>
<td>380 POKE -16368,0</td>
<td>Clear keyboard strobe - always after reading keyboard.</td>
</tr>
<tr>
<td>C061</td>
<td>390 X=PEEK(16287)</td>
<td>Read PDL(0) push button switch. If X&gt;127 then switch is &quot;on&quot;.</td>
</tr>
<tr>
<td>C062</td>
<td>400 X=PEEK(-16286)</td>
<td>Read PDL(1) push button switch.</td>
</tr>
<tr>
<td>C063</td>
<td>410 X=PEEK(-16285)</td>
<td>Read PDL(2) push button switch.</td>
</tr>
<tr>
<td>C058</td>
<td>420 POKE -16296,0</td>
<td>Clear Game I/O AN0 output</td>
</tr>
<tr>
<td>C059</td>
<td>430 POKE -16295,0</td>
<td>Set Game I/O AN0 output</td>
</tr>
<tr>
<td>C05A</td>
<td>440 POKE -16294,0</td>
<td>Clear Game I/O AN1 output</td>
</tr>
<tr>
<td>C05B</td>
<td>450 POKE -16293,0</td>
<td>Set Game I/O AN1 output</td>
</tr>
<tr>
<td>C05C</td>
<td>460 POKE -16292,0</td>
<td>Clear Game I/O AN2 output</td>
</tr>
<tr>
<td>C05D</td>
<td>470 POKE -16291,0</td>
<td>Set Game I/O AN2 output</td>
</tr>
<tr>
<td>C05E</td>
<td>480 POKE -16290,0</td>
<td>Clear Game I/O AN3 output</td>
</tr>
<tr>
<td>C05F</td>
<td>490 POKE -16289,0</td>
<td>Set Game I/O AN3 output</td>
</tr>
</tbody>
</table>
**APPLE II SYSTEM MONITOR**

**COPYRIGHT 1977 BY APPLE COMPUTER, INC.**

**ALL RIGHTS RESERVED**

S. WOZNIAK
A. BAUM

---

**TITLE**

```
LOC0  EP2  $00
LOC1  EP2  $01
WNDLFT EP2  $20
WNDWTH EP2  $21
WNDTOP EP2  $22
GDCTFM EP2  $23
CH   EP2  $24
CV   EP2  $25
GEASL EP2  $26
GEASH EP2  $27
OASL EP2  $28
OASH EP2  $29
OAS2L EP2  $2A
OAS2H EP2  $2B
H2   EP2  $2C
LANEM EP2  $2D
RTNL EP2  $2E
V2   EP2  $2F
RNEM EP2  $30
RNHH EP2  $31
MASK EP2  $32
CHRSUM EP2  $33
FORMAT EP2  $34
LASTIN EP2  $35
LENGTH EP2  $36
SIGN EP2  $3F
COLOR EP2  $40
MODE EP2  $41
LNPFLG EP2  $42
PROMPT EP2  $43
YSAV EP2  $44
YSAVI EP2  $45
CSWL EP2  $46
CSWH EP2  $47
KSWL EP2  $48
KSWH EP2  $49
PCL EP2  $4A
PCI EP2  $4B
XQT EP2  $4C
ALL EP2  $4D
ALH EP2  $4E
AIL EP2  $4F
ACH EP2  $50
AIL EP2  $51
AJB EP2  $52
AJL EP2  $53
AJR EP2  $54
AJF EP2  $55
ASF EP2  $56
ASR EP2  $57
ASL EP2  $58
ASH EP2  $59
```
FADE: A9 00 STA #ACC/256
FAKE: 85 41 STA #AR
FAP1: A2 FB LDX #SPB
FA54: A9 A0 RDSPI STA #SAO
FA55: 20 6D FD JSR COUT
FA55: BD 15 FA LDA RTBE=SPB,X
FA5C: 20 ED FD JSR COUT
FA5F: A9 BD LDA #$9D
FA61: 20 6D FD JSR COUT
FA64: B5 4A LDA ACC+$X
FA66: 20 DA FD JSR PRIVTE
FA69: E8 INX
FAFA: 30 E8 BMI RDSPI
FAPC: 60 RTS
FAPE: 18 BRANCH CLC BRANCH TAKEN.
FAPE: A0 01 LDX #$01 ADD LEN+2 TO PC
FA00: B1 3A LDA ($PCL),Y
FA02: 20 56 F9 JSR PCADJ3
FA05: 85 3A STA PCL
FA0C: 98 TYA
FA0E: 38 SEC
FA0F: B0 A2 BCS PCINC2
FA0F: 20 4A FF BRANCH JSR SAVE NORMAL RETURN AFTER
FA0F: 38 SEC XEO USER OF
FA0F: 90 9E BCS PCINC3 GO UPDATE PC
FAB0: EA INITSL NOP
FAB2: EA NOP DUMMY FILL FOR
FAB3: 4C 08 FR JMP BRANCH XEO AREA
FAB3: 4C FD 9A JMP BRANCH
FAB7: C1 RTBL DFB $C1
FAB1: D8 DFB $08
FAB1: D9 DFB $09
FAB3: 9D DFB $0D
FAB3: D3 DFB $03
FAB3: AD 70 C0 READ LDA PTMRG TRIGGER PADDLES
FAB9: A0 00 LDX #$00 INIT COUNT
FAB3: EA NOP COMPENSATE FOR 1ST COUNT
FAB5: EA NOP
FAB5: BD 64 C0 READ2 LDA PADDLO,X COUNT-Y=REG EVERY
FAB8: 10 04 BPL RTS2 12 USMP
FAB9: C8 INY
FABF: D0 98 BNE READ2 EXIT AT 255 MAX
FABD: 98 DEY
FABE: 60 RTS2B RTS
FABF: A9 00 INIT LDA #$00 CLR STATUS FOR DEBUG
FABF: 85 40sta STATUS SOFTWARE
FABF: AD 56 C0 LDA LORES
FABF: 9D 64 C0 LMPSCR INIT VIDEO MODE
FABB: AD 51 C0 SETTXT LDA TXTSET SET FOR TEXT MODE
FABF: A9 00 LDA #$00 FULL SCREEN WINDOW
FBC3: F0 08 BPL SETWD
FBC4: AD 50 C0 SETG4 LDA TXTCLP SET FOR GRAPHICS MODE
FBC4: AD 53 C0 LDA WIXSET LONER 4 LINES AS
FBC4: 20 36 FD JAR CLIPTOP TEXT WINDOW
FBC9: A9 14 LDA #$14
FBC9: 85 22 SETWD STA WNDTOP SET FOR 40 COL WINDOW
FBC9: A9 00 LDA #$00 TOP IN A-FPC.
FBC9: 85 20 WNDLT STA WNDLT @TTM AT LINE 24
FBCB: A9 28 LDA #$28
FBCB: 85 21 STA WNDWIDTH
FBCF: A9 18 LDA #$15
FBCF: 85 23 STA WNDTH VTAB TO ROW 23
FBCF: A9 17 LDA #$17
FBCF: 85 25 TABV STA CV VTAB TO ROW IN A-REC
FBCF: 4C 22 FC JMP VTAB
FBD4: 20 4A px MUPA JSR #01 ABS VAL OF AC AUX
FBD6: A0 10 MUL LDX #$10 INDEX FOR 16 BITS
FBD6: A5 50 MUL2 LDA $CL ACX+AXH+XINO
FBD7: 4A LSR A TO AC, XINO
FBD8: 90 0C ACC $M14 IF NO CARRY
FBD8: 18 CLC 4O PAPIAL PROD.
FBD9: A2 PE LDX #$FE
FBD9: A5 54 MUL3 LDA XTHDL+2,X AND UPLCN (AUX)
FBD9: 75 56 AUX MULX+2,X TO PAPIAL PROD
FBD9: 95 54 STA XTHDL+2,X (XTDN).
FBD7: EH INX
FBD7: D0 87 ANE $M13
FBD7: A2 03 MUL4 LEX #$03
FBD7: 76 MUL5 DFB #$76
FBD7: 50 CPF #$50
FBD7: CA DEY
FBD7: 10 PB BPL MUL5
FBD7: 88 DEY
FBD7: D0 85 BAE MUL2
FBD7: 60 PES
PF70: 20 C7 FF JSR G400E CLEAR VOM MODE, SCAN IDX
PF73: 20 A7 FF NXTITM JSR G610M GET ITEM, NON-HEX
PF76: 84 34 STY YSAV CHAR IN A-REG
PF78: AD 17 LDX #17 X-DEC=0 IF NO HEX INPUT
PF7A: 86 CHRSACHR DEF
PF7B: 30 E8 RTI VOM NOT FOUND, GO TO MAIN
PF7D: 09 CC FF CPF CHAR, THEN CMD CHAR IN TEL
PF80: D0 B6 CPF CHAR, THEN CMD CHAR IN TEL
PF81: 20 BE FF JSR T5009 FOUND, CALL CORRESPONDING
PF85: A4 34 LDX YSAV SUBROUTINE
PF8A: A2 03 DIC LDX #5
PF8C: OA 0A ASL A
PF8D: OA AO ASL A GOT HEX DIC,
PF8E: OA AO ASL A SHIFT INTO A2
PF8F: OA AO ASL A
PF90: OA AO ASL A
PF91: 26 35 ROL A2L
PF93: 26 3F ROL A2H
PF95: CA 0A DEX
PF96: 10 FA SPF NXTITM
PF98: A5 31 NXTMAS LDA #128
PF9A: D0 B6 SPF NXTMAS2 IF NOM IS 0,0
PF9C: 85 3F LDA A2H,X THEN COPY A2 TO
PF9E: 95 0D STA A1Y,X A1 AND A3
PFAD: 95 01 STA A3,X
PFAD: D8 05 ROL 
PA3: D0 06 SPF NXTCHAR
PA4: A2 00 OPSVY LDA #30 CLFA A2
PAA: 86 3E ROL A2L
PB1: 86 3F ROL A2H
PBA: E9 00 02 VXTPCP LDA INY, Y GET CHAR
PB3: C8 4C CPY
PB4: 49 0A SPF 52U
PB5: C9 0A CMF 52A
PB6: 90 D3 SCC DIC IF HEX DIC, THEN
PB67: 69 88 ADC #5F8
PB9: C9 FA CMF #5FA
PB91: 00 00 SCC #10
PB9D: 60 09 RET
PB9D: 60 09 RET
PB9E: A9 0E 10SUG LDA (0C/256) LOW-HIGH-OVER
PC0: 48 PRP SUPP ADV ON REG
PC1: 69 E3 FF LCA CMD, Y, PROF IN ORDER
PC4: A6 40 PRP SUPP ADV ON REG
PC5: A5 31 LDA #5B
PC7: A9 00 3100M LFX $100 CLP MOVE, CMD MODE
PC9: D4 31 LFX $100 TO A-REG
PCB: 60 65 GTS GO TO SUBR VIA PTE
PDA: 95 CNTPUL DPP $3C "CTRL-C"
PDA: 0B DPP $29 "CTRL-Y"
PCE: B6 DPP $2A "CTRL-X"
PCC: BE DPP $2B "CTRL-Z"
PCE: CF DPP $50 "F(""
PCE: C0 DPP $51 "F(""
PCE: C1 DPP $52 "F(""
PCE: C2 DPP $53 "F(""
PCE: C3 DPP $54 "F(""
PCE: C4 DPP $55 "F(""
PCE: C5 DPP $56 "F(""
PCE: C6 DPP $57 "F(""
PCE: C7 DPP $3A "F(""
PCE: D0 DPP $37 "F(""
PCE: D1 DPP $38 "F(""
PCE: D2 DPP $39 "F(""
PCE: D3 DPP $3A "F(""
PCE: D4 DPP $3B "F(""
PCE: D5 DPP $3C "F(""
PCE: D6 DPP $3D "F(""
PCE: D7 DPP $3E "F(""
PCE: D8 DPP $3F "F(""
PCE: E1 DPP $3A E1 (F=EX-OP SE0$89)
PCE: E2 DPP $3B E2 (F=EX-OP SE0$89)
PCE: E3 DPP $3C E3 (F=EX-OP SE0$89)
PCE: E4 DPP $3D E4 (F=EX-OP SE0$89)
PCE: E5 DPP $3E E5 (F=EX-OP SE0$89)
PCE: E6 DPP $3F E6 (F=EX-OP SE0$89)
PCE: F1 DPP $3F F1 (F=EX-OP SE0$89)
PCE: F2 DPP $3F F2 (F=EX-OP SE0$89)
PCE: F3 DPP $3F F3 (F=EX-OP SE0$89)
PCE: F4 DPP $3F F4 (F=EX-OP SE0$89)
PCE: F5 DPP $3F F5 (F=EX-OP SE0$89)
PCE: F6 DPP $3F F6 (F=EX-OP SE0$89)
PCE: F7 DPP $3F F7 (F=EX-OP SE0$89)
PCE: F8 DPP $3F F8 (F=EX-OP SE0$89)
PCE: F9 DPP $3F F9 (F=EX-OP SE0$89)
PCE: FA DPP $3F FA (F=EX-OP SE0$89)
PCE: FB DPP $3F FB (F=EX-OP SE0$89)
PCE: FC DPP $3F FC (F=EX-OP SE0$89)
PCE: FD DPP $3F FD (F=EX-OP SE0$89)
PCE: FE DPP $3F FE (F=EX-OP SE0$89)
PCE: FF DPP $3F FF (F=EX-OP SE0$89)
FFF0: B3  DFB  #SET:CRM-1
FFF1: 76  DFB  #SET:INV-1
FFF2: 50  DFB  #LIST-1
FFF3: CC  DFB  #TRIE-1
FFF4: 45  DFB  #GO-1
FFF5: FC  DFB  #READ-1
FFF6: 17  DFB  #SET:MODE-1
FFF7: 17  DFB  #SET:MODE-1
FFF8: F3  VFA  #CHR:ON-1
FFF9: 03  DFB  #BLANK-1
FFFA: F3  DFB  #MHI  MHI VECTOR
FFFF: 03  DFB  #MHI/256
FEFC: 59  DFB  #RESET  RESET VECTOR
FFFD: FF  DFB  #RESET/256
FFFE: 66  DFB  #IRC  IRC VECTOR
FFFF: F3  DFB  #IRC/256

KOTIC  EOU  32C
***************

APPLE-II

MINI-ASSEMBLER

COPYRIGHT 1977 BY
APPLE COMPUTER INC.

ALL RIGHTS RESERVED

S. NOZNIK

A. ZAUN

***************

TITLE "APPLE-II MINI-ASSEMBLER"

FORMAT EPY 92F
LENGTH 92F
CODE EPY 911
PROMPT EPY 913
VSAV EPY 914
C EPY 92B
PCL EPY 93A
PCH EPY 93C
ALU RK 93D
A2L EPY 93I
A2L EPY 93F
A2L RK 942
A41 RK 943
PST EPY 949
L EPY 9200
IN0065 RK 9067
IN0063 RK 9060
PRL12 RK 904A
PRL14 RK 9053
CHAR1 RK 9104
CHAR2 RK 9074
CHGP RK 90C0
CHMP RK 9069
CURSUP RK 9C1A
CURLZ RK 9067
CURT RK 90E0
MLI RK 9C09
MCLP RK 9C1A
PULL RK 9F3A
GFHUA RK 9FFA
GBO RK 9FFC
3ONE RK 9FFCC

$500: CN 41 RELZ DEX
$502: 4A DEX
$503: D0 14 DEX
$505: A4 3F DEX
$507: A6 3F DEX
$509: 00 01 RELZ DEX
$50B: 88 DEX
$50C: CA RELZ DEX
$50D: 8A TXA
$50E: 18 CLC
$50F: 3A SRC DLC FORM ADDR-PG-2
$511: 85 3E SRC A2L
$513: 10 01 SRC PREL3
$515: C8 INY
$516: 98 REL3 TXA

91
**F517**: E3 39  
**F519**: D0 68  
**F51B**: A4 2F  
**F51D**: E9 F0  
**F520**: 93 3A  
**F522**: 88  
**F523**: 00  
**F525**: 20 1A  
**F526**: 00 1A  
**F528**: 20 00  
**F52A**: 00 53  
**F531**: 88 3B  
**F533**: 7A  
**F535**: 4C 95  
**F53B**: 20 2E  
**F53D**: 20 07  
**F540**: 84 34  
**F542**: A0 17  
**F544**: 00  
**F545**: 4C 95  
**F549**: 20 00  
**F54B**: A3 95  
**F551**: 20 0E  
**F555**: 00  
**F557**: 00  
**F559**: 4C 95  
**F55B**: 20 0E  
**F561**: AA  
**F563**: BB 00  
**F565**: C5 42  
**F567**: DD 1A  
**F569**: BD 00  
**F56B**: C5 43  
**F56D**: DD 00  
**F570**: A5 1A  
**F572**: AD 2E  
**F574**: C0 9D  
**F576**: 00 B0  
**F578**: C5 2E  
**F57A**: 00  
**F57C**: C6 3D  
**F57E**: DD 0C  
**F580**: E6 44  
**F582**: C6 35  
**F584**: 00 D0  
**F586**: A4 34  
**F588**: 99  
**F589**: AA  
**F58B**: 20 4A  
**F58D**: A9 DE  
**F58F**: 00  
**F592**: 20 3A  
**F595**: A9 A1  
**F597**: B3 33  
**F599**: 20 67  
**F59C**: 20 C7  
**F59F**: AD 00  
**F5A2**: CA 90  
**F5A4**: F0 33  
**F5A6**: E8  
**F5A8**: C9 A4  
**F5A9**: F0 92  
**F5AC**: 20 A7  
**F5AF**: C9 93  
**F5B1**: D0 05  
**F5B3**: 0A  
**F5B4**: F0 D2  
**F5B6**: 20 76  
**F5B8**: A9 03  
**F5BA**: E8 35  
**F5BE**: 20 34  
**F5C0**: 0A  
**F5C1**: E9 BE  
**F5C3**: C9 C2  
**F5C5**: 90 C1  
**F5C7**: 0A  
**F5C8**: 0A  
**F5C9**: A2 04  
**F5CB**: 0A

**Error if >1-byte branch**

**Error if unrecognised delin**

**Check next delin**

**Handle CR outside monitor**

**Get trial opcode**

**Get trial format**

**Get trial format relative?**

**Get trial format same format?**

**Find op yes**

**Find op next opcode**

**Print "under last read char" to indicate error**

**Print "in col 1?**

**Yes, simulate monitor**

**No, backup a char**

**JSP GEPMKM GET A NUMBER**

**JSP GEPMKM GET TERMINATOR?**

**Move aod preceding color**

**Count of chars in mnemonic**

**Move first mnew char.**

**Extract offset**

**Legal char?**

**No.**

**Compress - left justify**

**Do 5 triplet word shifts**
P5CC:  26 42  ROL  A8H
P5C6:  26 43  ROL  A4H
P5DD0:  CA  DEY
P5D1:  10  F8  BPL  NWTM2
P5D3:  C6  3D  DEC  A1H  DONE WITH 3 CHAR?
P5D4:  F0  F4  BFO  NWTM2  YES, BUT NO 1 MORE SHIFT
P5D7:  10  E4  BPL  NWTM4  NO
P5D9:  A2  05  FORM1  LDX  $E5  5 CHARs IN ADDP MODE
P5D8:  20  34  P6  FORM2  JSR  GETMSp  GET FIRST CHAR OF ADDR
P5DB:  84  34  STY  YSW
P5ED:  D0  04  P9  CMP  CHAR4,X  FIRST CHAR MATCH PATTERN?
P5E3:  D0  13  BNE  FORM3  NO
P5E5:  20  34  P6  JSR  GETMSp  YES, GET SECOND CHAR
P5ED:  05  C2  P9  CMP  CHAR2,X  MATCHED SECOND HALF?
P5EF:  F0  0D  BNE  FORM5  YES
P5E0:  80  0A  P9  LDA  CHAR2,X  NO, IS SBCUS0 HALF ZERO?
P5E4:  BF  U7  BEQ  FORM4  YES.
P5E2:  CA  A4  CMP  $0A  NO, SBCUS0 HALF Optional?
P5E8:  F0  03  BNE  FORM4  YES.
P5E9:  A4  34  LCY  YSW
P5E8:  10  FORM3  CLC  CLEAR BIT-NO MATCH
P5E9:  80  FORM4  DEX  BACK UP 1 CHAR
P5EX:  26  44  FORM5  ROL  FAH  FORY FORMAT BYTE
P5EC:  E0  03  CFP  #$3  TIME TO CHECK FOR ADDR.
P5EF:  D0  00  BNL  FORM7  NO
P5EH:  20  A7  PP  JSR  GETMN0  YES
P603:  N5  3F  LDA  #$2
P605:  F0  01  BEQ  FORM6  HIGH-GROUP BYTE ZERO
P607:  E7  INX  NO, INC OR FOR 2-BYTE
P608:  B6  35  FORM6  STX  L  STORP LENGTH
P60A:  A2  03  LDX  #$3  PLOAD FORMAT INDEX
P60C:  B0  DEY  BACKUP A CHAR
P60D:  96  3D  FORM7  STX  A1H  SAVE INDEX
P60F:  CA  DEY  DONE WITH FORMAT CHECK?
P610:  10  C9  BPL  FORM2  NO.
P612:  A5  44  LDA  FMT  YES, PUT LENGTH
P614:  0A  ASL  A  IN LOW RITEs
P615:  DA  ASL  A
P616:  05  35  ORA  L
P618:  C9  20  CMP  #$20
P61A:  B0  06  PCS  FORM8  ADD 'S' IF NONZERO LENGTH
P61C:  A6  35  LDX  L  AND DON'T ALREADY HAVE IT
P61D:  F0  02  BEQ  FORM8
P620:  09  80  ORA  #$0  $00
P622:  85  44  STA  FMT
P624:  84  34  STY  YSW
P626:  89  00  02  LDA  IN.Y  GET NEXT NONBLANK
P629:  C9  B8  CMP  #$89  ';' START OF COMMENT?
P62B:  F0  04  BEQ  FORM9  YES
P62D:  C9  80  CMP  #$8H  CAPRIAGE RETURN?
P62F:  D0  00  BNE  BRK4  NO, BRK.
P631:  4C  5C  P5  FORM9  JMP  TRYNEXT
P634:  B9  00  02  GETMSp  LDA  IN.Y
P637:  C8  INY
P638:  C9  A0  CMP  #$A0  GET NEXT NON BLANK CHAR
P63A:  F0  F8  BEQ  GETMSp
P63C:  60  RTS
P666:  4C  92  F5  MINASH  JMP  RESET2
```

***********************
*                  *
*  APPLE-II FLOATING  *
*  POINT ROUTINES   *
*                  *
* COPYRIGHT 1977 BY *
*  APPLE COMPUTER INC. *
*                  *
*  ALL RIGHTS RESERVED *
*                 *
*  S. KOZNIK          *
*                  *
***********************

TITLE "FLOATING POINT ROUTINES"

SIGN          SF  SF
XZ            EP  EP
A2            SF  SF
X1            EP  EP
E             EP  EP
LWLOC         CNU S3F5
SOM          SF423
ADD          CLC
           
F425: 1A
F426: A2 02
F428: B5 F9
F42A: 75 F5
F42C: 95 F9
F42E: CA
F42F: 10 F7
F431: 60
F432: 06 F3
F434: 20 37 F4
F437: 24 F9
F439: 10 05
F43A: 20 A4 F8
F43B: B6 F3
F440: 88
F441: A7 04
F443: 84 F8
F445: 63 F7
F447: 04 F3
F449: 94 F7
F44B: 95 F3
F44D: CA
F44E: 0D F3
F450: 60
F451: A5 F8
F453: SS F8
F455: A5 F9
F457: C9 C0
F459: 30 0C
F45C: 0F F8
F45E: 06 F8
F460: 2C FA
F461: 26 F9
F463: A5 F8
F465: D0 EE
F467: 60
F468: 20 A4 F4
F46A: 20 7B F4
F46C: A5 F4
F470: C5 F8
F472: 0D F7
F474: 20 25 F4
F476: 50 EA
F478: 7D 05

CLC          CLEAR CARRY.
IDX #67 INDEX FOR 3-BYTE ADD.
LDA #1,X ADD A TO MANT2 OR MANT1.
STA #1,X ADD A TO MANT2 OR MANT1.
DEX INDEX TO NEXT MORE SIGNIF. BYTE.
BPL ADD1 LOOP UNTIL DONE.
RTS RETURN

A61 SIGN CLEAR USE OF SIGN.
JR 437

ABSADF STX #1 MANT NEGATIVE?
LVL ABSADF1 LDY, SWAP WITH MANT2 AND RETURN.
JSR PCOMP YES, COMPLEMENT IT.
INC SIGN INC SIGN, COMPLEMENTING LSB.

ABSADF SRC SET CARRY FOR RETURN TO MUL/DIV.

INDEX FOR 2-BYTE SWAP.
LDA X1-1,X SWAP A BYTE OF EXP/MANT WITH
LDB X2-1,X EXP/MANT1 AND LEAVE A COPY OF
STA Y2-1,X MANT1 IN E (3 BYTES). F+3 USED

DFX ADVANCE INDEX TO NEXT BYTE.

BPL SWAP1 LOOP UNTIL DONE.

RETURN

INIT EXP1 TO 14,
THEN NORMALIZE TO FLOAT.
HIGH-ORDER MANT1 BYTE.
UPPER TO BITS UNREAL?
YEN, RETURN WITH MANT1 NORMALIZED
DEC X1 DECREMENT EXP1.

ACL #1+2

SHIFT MANT1 (3 BYTES) LEFT.

EXPI SER0?

NO, CONTINUE NORMALIZING.

RTS1 RTS

COMARE EXP1 WITH EXP2.

IF 1, SWAP ADDEND OR ALIGN MANTS.

ADC ALIGNED MANTISSAS.

NO OVERFLOW, NORMALIZE RESULT.

MUL MANT2 RIGHT, CARRY INTO SIGN

94
```
F47B: 90 C4 ALONG A SWAP
     100 C4 SWAP RIGHT ARITH.*
F47D: A5 P9 STAP
       0A LDA "1" SIGN OF MANT1 INTO CARRY FOR
F47E: DA 74 SAA A FIGHT AGAINST SHIFT.
F480: E6 F8 XFD
       3C INC X1 TO ADJUST FOR RIGHT SHIFT.
F482: F0 75 ECP OVF1
       EXPL OUT OF RANGE.
F484: A2 FA 120G1 LDX #5FA INDEX FOR 64-65B RIGHT SHIFT.
F486: 72 F8 NEP1 FOR #1 X
F488: E8 NH4 NEXT BYTE OF SHIFT.
F489: D0 P8 CNE 0A1 LOOP UNTIL DONE.
F488: 60 FTS RETURN.
F48C: 20 32 F4 FUL
       JSR #1 AND VAL OF MANT1, MANT2.
F490: 18 C4 CMP X1 AND EXP1 TO EXP2 FOR PRODUCT EXP.
        01 AOC X1
F491: 20 E2 F4 JSR #02 CHECK PROD. EXP AND PREP. FOR MUL.
F494: 18 CLC CLEAR CARRY FOR FIRST BIT.
F495: 20 84 F4 MUL1 JSR 120G1 16 ADD OF PRODUCT (PROD AND MUL1).
F496: 90 02 JSR #02 IF CARRY CLEAR, SKIP PARTIAL PROD.
F497: 20 25 F4 JSR #00 AND MULTIPICAND TO PRODUCT.
F49D: 18 MUL2 CPY NEXT "1" ITERATION.
F49E: 10 F5 MUL3 LOOP UNTIL DONE.
F49F: 46 F3 MUL3
       JSR SET SIGN.
F4A0: 90 27 MUL4 JSR #27 IF EVEN, NORMALIZE PROD, ELSE COMP.
F4A4: 38 SEC SET CARRY FOR SUBTRACT.
F4A5: A2 03 LDX #3 INDEX FOR 3-BYTE SUBTRACT.
F4A7: A9 00 C04PL
       LDA #50 CLEAR A.
F4A8: 85 F8 SCP X1,X SUBTRACT BYTE OF EXP1.
F4AD: CA DEX NEXT MORE SIGNIFICANT BYTE.
F4AE: D0 P7 BNE COMPL1 LOOP UNTIL DONE.
F4BF: F0 C5 SEQ ADDENDO NORMALIZE (OR SHIFT RT IF OVF1).
F4BG: 12 F4 PDIV JSR #02 TAKE ABS VALUES OF MANT1, MANT2.
F4BD: F5 P0 SPC X1 SUBTRACT EXP1 FROM EXP2.
F4B7: 20 E2 F4 JSR #02 SAVE AS QUOTIENT EXP.
F4BA: 38 DIV1 SEC SET CARRY FOR SUBTRACT.
F4BB: 42 02 LEX #52 INDEX FOR 3-BYTE SUBTRACTION.
F4BE: 85 LD3 DIV2 LDA #2,Y SUBTRACT A BYTE OF E FROM MANT2.
F4BF: F5 PC SEC E,Y SAVE O1 STACK.
F4C1: 48 PHA NEXT HOEF SIGNIFICANT BYTE.
F4C2: CA DEX NEXT HOEF SIGNIFICANT BYTE.
F4C3: 18 L0 P8 APL PIV2 INDEX FOR 3-BYTE CONDITIONAL MOVE.
F4C5: A2 FD LDX #FD LOAD INDEX FOR DIFFERENCE OF STACK.
F4C7: 08 DIV3 PLA FULL BYTE OF DIFFERENCE OF STACK.
F4CB: 90 02 BCC DIV4 IF MZ/KE THEN DON'T RESTORE MZ.
F4CA: 95 P8 STA #2,X SAVE DIV3.
F4CC: E8 DIV4 INX JSR LESS SIGNIFICANT BYTE.
F4CD: D0 P8 DNE DIV3 LOOP UNTIL DONE.
F4CF: 26 F0 ROL #1,X ROLL QUOTIENT LEFT, CARRY INTO LSA.
F4D1: 26 FA ROL #1,Y ROLL QUOTIENT INTO LSR.
F4D3: 26 P9 ROL #1,Z ROLL QUOTIENT INTO LSH.
F4D5: 06 P7 ASL #2,Z SHIFT DIVIDEND LEFT.
F4D7: 26 FD ROL #2,Z
F4DB: 80 A1 LCA OVS.
F4D8: B0 1C NCS OVL.
F4D9: 29 P5 ROL #2.
F4DA: B0 9C OVS.
F4DB: 00 DA DEL.
F4DE: 00 DA RNE DIV1
F4E0: F0 BE RNE DIV1.
F4E5: 86 FA STX #1
F4E6: 86 FA STA #1
F4E8: 80 00 ACS OVC3.
F4EA: 30 04 M4 M3
F4EC: 68 PLA POP ONE RETURN LEVEL.
F4ED: 68 PLA
F4EE: 90 B2 ACC #CHR.
F4F0: 49 B0 #3 FOR #580 COMPLEMENT SIGN BIT OF EXPONENT.
F4F2: 85 F8 STA X1 STORE IT.
F4F4: A0 17 LOY #17 COUNT 24 64/23 DIV ITERATIONS.
F4F6: 60 PIS RETURN.
F4F7: 10 F7 OVC1
       XPL #03 IF POSITIVE EXP THEN NO OVL.
F4F9: 4C P5 03 OVEL JAP UVLOC
       INC #03.
F4F9: 0D F0 00 OVEL JAP UVLOC
       INC #03.
F4FA: 20 70 F4 FIX JSR STAR
F4FB: A5 P8 FIX LDA #1
F4FE: D1 13 FIX U+1
F4FB: C9 8E CAP #58.
F4FF: D0 53 GRT FIX1
F504: 29 F9 BIT #1
F506: A4 10 OA MFL FIX1.
F50C: A5 P8 LDA #1,X
F50E: F0 06 PFO FIX4.
F510: 85 FA INC #1,Y
F512: D0 02 MFL FIX4.
F514: E6 F9 INC #1,Z
F516: 60 FIXRTS RTS.
F517: A3 00 UVFL LDA #50.
F51B: 80 P9 STA #1,
F51B: 85 FA STA #1,
F51D: 60 FTS
**APPLE-II PSEUDO**
**MACHINE INTERPRETER**
**COPYRIGHT 1977**
**APPLE COMPUTER INC.**
**ALL RIGHTS RESERVED**

**S. KOZNIK**

**--------------------**

**TITLE "SWEET16 INTERPRETER"**

```
SOL EPZ $0
ROH EPZ $1
R14d EPZ $10
S16PAG EQU $57
SAVE EQU $FF4A
RESTORE EQU $FF3F
```
F601: 95 01 STA R0H,X
F603: 96 DEY
F604: 01 1E LDA (H:15),Y LOW-ORDER BYTE OF CONSTANT
F606: 45 90 STA ROL,X
F608: 9B TYA Y-REG CONTAINS 1
F609: 3E SFC
F60A: 65 1F ADC R1L ROL 2 TO R0C
F60C: 85 1F STA R15,X
F60E: 90 02 BCC SRET2
F610: E6 1F INC R13
F612: 60 SFR2 R2S
F613: 02 ORPH BRPL.D FFF8 SET-1 (1X)
F614: F9 BRPL.D FFF8 SET-1 (0)
F615: 04 DFH LD-1 (7X)
F616: 9D DFH CA-1 (1)
F617: 0D DFH ST-1 (3X)
F618: 9E DFH MC-1 (2)
F619: 25 DFH [DAT-1 (4X)
F61A: AF DFH SC-1 (3)
F61B: 16 DFH STAP-1 (5X)
F61C: B2 DFH ER-1 (4)
F61D: 47 DFH LDDAT-1 (6X)
F61E: 09 DFH MS-1 (5)
F61F: 51 DFH STNAT-1 (7X)
F620: C0 DFH PZ-1 (6)
F621: 2F DFH POP-1 (8X)
F622: C9 DFH AN1-1 (7)
F623: 5E DFH STAP-1 (9X)
F624: D2 DFH SM-1 (8)
F625: 85 DFH ADD-1 (AX)
F626: D0 DFH ANM1-1 (9)
F627: 6E DFH SUB-1 (EX)
F628: 05 DFH N-1 (A)
F629: 31 DFH MORP-1 (CX)
F62A: E8 DFH PS-1 (E)
F62B: 70 DFH CCR-1 (DX)
F62C: 93 DFH BS-1 (C)
F62D: 1E DFH INR-1 (FX)
F62E: E7 DFH MUL-1 (D)
F62F: 65 DFH DCR-1 (FX)
F700: E7 DFH NUL-1 (E)
F701: E7 DFH MUL1-1 (UNEUSED)
F702: E7 DFH NUL-1 (F)
F703: 10 CA SET DFH SRET2 ALWAYS TAKEN
F705: 05 00 LD ROL,X
F707: 85 00 STA R0L
F709: B5 02 LDA R0M,X MOVE RX TO A0
F70B: 85 01 STA R0M
F70D: 60 RTS
F70E: A5 00 ST LDA R0L
F710: 95 00 STA ROL,X MOVE R0 TO RX
F712: A5 01 LDA R0H
F714: 95 01 STA R0H,X
F716: 60 RTS
F717: A5 00 STAT LDA ROL
F719: A1 00 STAT2 STA (R0L,X) STORE BYTE INDIRECT
F71B: A0 00 LDY $60
F71D: 84 1D STAT3 STY $14H INDICATE R0 IS RESULT REG
F71F: F6 00 INR INC ROL,X
F721: D0 02 SNE INR2 INCR RX
F722: F6 01 INC ROL,X
F725: 60 RTS INR2 RTS
F726: A1 00 LDAT LDA (R0L,X) LOAD INDIRECT (RX)
F728: 85 00 STA ROL 10 RO
F72A: A0 00 LDY $50
F72C: 84 01 STY R0H 01 ZERO HIGH-ORDER PO BYTE
F72E: F0 ED BEH R0H3 ALWAYS TAKEN
F730: A0 00 POP LDY $50 HIGH ORDER BYTE = 0
F732: F0 06 BEH POP2 ALWAYS TAKEN
F734: 26 66 0F7 POPD JSR DCP DEC RX
F737: A1 00 LDA (R0L,X) POP HIGH-ORDER BYTE RX
F739: A8 TAY SAVE IN Y-REG
F73A: 20 66 0F7 POL2 JSR DCP DEC RX
F73D: A1 00 LDA (R0L,X) LOW-ORDER BYTE
F73F: 85 00 STA R0L TO RO
F741: 84 01 STY R0H
F743: A0 00 POP3 LDY $50 INDICATE RO AS LAST RST REG
F745: 84 1D STY $14H
F747: 60 RTS
F748: 20 26 0F7 LDA ROL,D JKP LDAT LOW-ORDER BYTE TO RX, INCR RX
F748: A1 00 LDA (R0L,X) HIGH-ORDER BYTE TO RX
F74A: 85 01 STA R0M
F74F: 4C 1F JMP INCR RX
F752: 20 17 0F7 STAP JSR SYMT STORE INDIRECT LOW-ORDER
F755: A5 01   LDA #0D   BYTE AND INCR RX, THEN
F757: B1 00   STA (R0.L,X) STORE HIGH-ORDER BYTE.
F759: 4C 1F F7   JMP INCX INCR RX AND RETURN
F75C: 20 B6 F7   STPA T JSP DCF DECP FX
F75F: A5 00   LDA #0E   LOAD #0E AS LAST ASLI REG
F761: 81 00   STA (R0.L,X) STORE RX LO\(W\) BYTE OR X
F763: 4C 63 F7   JVP POH I7 INDICATE PO AS LAST ASLI REG
F766: B5 00   LDA #0F   LOAD #0F AS LAST ASLI REG
F768: D0 02   DAE BCP7 DECX FX
F76A: D6 01   DFC RXH,X
F76C: D6 00   DEX RXL,X
F76E: 60 RTE
F770: A9 00   SUBB #0F   DECL TO #00
F771: 38 CPR   SEC   NOTE Y-REG = 13#2 FOR CPR
F772: A5 00   LDA ROH
F774: F5 00   SBC RXH,X
F776: 99 00 00   STA RXH,Y   RD-RX TO PY
F779: A5 01   LDA ROH
F77B: F5 01   SBC ROH,X
F77D: 99 00 00   SUB2   STA ROH,Y
F780: 98 TAA   LAST RESULT REG#2
F781: D9 00   ADC #00   CARRY TO LSH
F783: E5 1D   STA RXH
F785: 60 RTS
F786: A5 00   ADD #00   LDA ROH
F788: 75 00   ADC ROH,X
F78A: B5 00   STA ROH   RX+RX TO #00
F78C: A5 01   LDA ROH
F78E: 75 01   ADD ROH,X
F790: 0A 00   LDA #00   RX FOR RESULT
F792: F0 E9   RED SUB2   FINISH ADD
F794: A5 1E B5   LDA #15L   NOTE X-REG IS 32#21
F796: 20 19 F7   JSR STA2   PUSH LOW PC BYTE VIA R12
F798: A5 1F   LDA #15H
F79B: 20 1F F7   JSR STA2   PUSH HIGH-ORDER PC BYTE
F79E: 18 BR   CLI
F79F: B0 0E   BNC   ECS #0C2   NO CARRY TEST
F7A1: B1 1E   BNL1   LDA (#15H),Y DISPLACEMENT BYTE
F7A3: 10 01   BPL #07
F7A5: 88 DEX
F7A6: 65 1E   BR2   ADD #15L   ADD TO PC
F7A8: 85 1E   STA #15L
F7AA: 98 TAY
F7AB: 65 1F   ADC #15H
F7AD: 85 1F   STA #15H
F7AF: 60 BNC2   RTS
F7B0: 80 EC   SC   PCS #0R
F7B2: 60 RTS
F7B3: 0A 3F   ASL A   LOUISE RESULT-REG INDEX
F7B4: AA TAX   TU X-REG FOR INDEXING
F7B5: B5 01   LDA ROH,X   TEST FOR PLUS
F7B7: 10 EE   EPL PPI   BRANCH IF SO
F7B9: 60 RTS
F7BA: 0A 9A   ASL A   LOUISE RESULT-REG INDEX
F7BB: AA TAX
F7BC: 85 00   LDA RXH,X   TEST FOR ZERO
F7BD: 15 01   ORA #00,X (BOTH BYTES)
F7BE: F0 08   DEC #1   BRANCH IF SO
F7C0: 60 RTS
F7C1: 0A 82   ASL A   DOUBLE RESULT-REG INDEX
F7C2: AA TAX
F7C3: B5 00   LIA RXH,X   TEST FOR ZEROS
F7C5: 15 01   ORA #00,X (BOTH BYTES)
F7C7: F0 06   DEC #1   BRANCH IF SO
F7C9: 60 RTS
F7CA: 0A 82X   ASL A   DOUBLE RESULT-REG INDEX
F7CB: AA TAX
F7C7: 05 00   LDA RXH,X   TEST FOR NONZEROS
F7CD: 15 01   ORA RXH,X (BOTH BYTES)
F7D0: D0 CF   BNE #01   BRANCH IF SO
F7D2: 60 RTS
F7D3: 0A BM1 ASL A   DOUBLE RESULT-REG INDEX
F7D4: AA TAX
F7D5: B5 00   LDA RXH,X   CHECK BOTH BYTES
F7D7: 35 01   AND RXH,X FOR SFR (MINUS 1)
F7D9: 49 FF   EOR #SFF
F7DB: F0 C4   DEC #1   BRANCH IF SO
F7DD: 60 RTS
F7DE: 0A BM1 ASL A   DOUBLE RESULT-REG INDEX
F7DF: AA TAX
F7E1: B5 00   LDA RXH,X   CHECK BOTH BYTES FOR NO SFF
F7E2: 35 01   AND RXH,X
F7E4: 49 FF   EOR #SFF
F7E6: 00 B9   BNE #01   BRANCH IF NOT MINUS 1
F7E8: 60 RTS
F7E9: A2 18   ASL RX   #118   12#2 FOR R12 AS STK POINTER
F7EB: 20 66 F7  JSR DCR  DEC R STACK POINTER
F7EE: A1 00  LDA (R0L,X)  POP HIGH RETURN ADDR TO PC
F7F0: B5 1F  STA #15H
F7F2: 20 66 F7  JSR DCS  SAME FOR LOW-ORDER BYTE
F7F5: A1 00  LDA (R0L,X)
F7F7: B5 1E  STA #15L
F7FA: 4C C7 F6 RTR  JMP RTRZ

6502 MICROPROCESSOR INSTRUCTIONS

AOC  Add Memory to Accumulator with Carry
AND  "AND" Memory with Accumulator
ASL  Shift Left One Bit (Memory or Accumulator)
BCC  Branch on Carry Clear
BCS  Branch on Carry Set
BED  Branch on Result Zero
BIT  Test Bits in Memory with Accumulator
BMI  Branch on Result Minus
ONE  Branch on Result not Zero
BPL  Branch on Result Plus
BRK  Force Break
BVC  Branch on Overflow Clear
BVS  Branch on Overflow Set
CLC  Clear Carry Flag
CLD  Clear Decimal Mode
CLI  Clear Interrupt Disable Bit
CLV  Clear Overflow Flag
CMP  Compare Memory and Accumulator
CPX  Compare Memory and Index X
CPY  Compare Memory and Index `I
DEC  Decrement Memory by One
DEX  Decrement index X by One
DEY  Decrement Index Y by One
FOR  "Exclusive-Or" Memory with Accumulator
INC  Increment Memory by One
INX  Increment Index X by One
INY  Increment Index `I by One
JMP  Jump to New Location
JSA  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 Right one Bit (Memory or Accumulator)
NOP  No Operation
ORA  OR Memory with Accumulator
PHA  Push Accumulator on Stack
PHP  Push Processor Status on Stack
PLA  Pull Accumulator from Stack
PLP  Pull Processor Status from Slack
ROL  Rotate One Bit Left (Memory or Accumulator)
ROR  Rotate One Bit Right (Memory or Accumulator)
RTI  Return from Interrupt
RTS  Return from Subroutine
SBC  Subtract Memory from Accumulator with Borrow
SEC  Set Carry Flag
SED  Set Decimal Mode
SEI  Set Interrupt Disable Status
STA  Store Accumulator in Memory
STX  Store Index X in Memory
STY  Store Index Y in Memory
TAX  Transfer Accumulator to Index X
TAY  Transfer Accumulator to Index Y
TSX  Transfer Stack Pointer to Index X
TXA  Transfer Index X to Accumulator
TXS  Transfer Index X to Stack Pointer
TYA  Transfer Index Y to Accumulator
THE FOLLOWING NOTATION APPLIES TO THIS SUMMARY:

FIGURE 1. ASL-SHIFT LEFT ONE BIT OPERATION

FIGURE 2. ROTATE ONE BIT LEFT (MEMORY OR ACCUMULATOR)

FIGURE 3.

NOTE 1: BIT — TEST BITS

Bit 6 and 7 are Iranatered to the status register. If the result of $A M$ is zero than $Z=1$, otherwise $Z=0$.

PROGRAMMING MODEL

ACCUMULATOR

INDEX REGISTER Y

INDEX REGISTER X

PROGRAM COUNTER

STACK POINTER

PROCESSOR STATUS REGISTER, "P"

CARRY
ZERO
INTERRUPT DISABLE
DECIMAL MODE
BREAK COMMAND
OVERFLOW
NEGATIVE
### INSTRUCTION CODES

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX OP Code</th>
<th>No. Bytes</th>
<th>'P' Status Reg.</th>
<th>N Z C I O V</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ADC</strong></td>
<td>Add memory to accumulator with carry</td>
<td>A.M.C — A.C</td>
<td>Immediate ADC #Op A</td>
<td>69 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page ADC Op A</td>
<td>62 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute ADC Op X</td>
<td>75 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, X ADC Op Y</td>
<td>60 3</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, Y (direct.X) ADC Op X</td>
<td>70 3</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, Y (indirect.Y) ADC Op Y</td>
<td>71 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>AND</strong></td>
<td>Subtract memory with accumulator</td>
<td>A.M — A</td>
<td>Immediate AND #Op A</td>
<td>29 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page AND Op A</td>
<td>25 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute AND Op X</td>
<td>35 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, X AND Op Y</td>
<td>20 3</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, Y AND Op X</td>
<td>39 3</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(direct.X) AND Op X</td>
<td>21 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(indirect.Y) AND Op Y</td>
<td>31 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>ASL</strong></td>
<td>Shift left one bit (Memory or Accumulator)</td>
<td>(See Figure 1)</td>
<td>Accumulator ASL A A</td>
<td>0A 1</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page ASL Op A</td>
<td>06 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, X ASL Op X</td>
<td>16 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, Y ASL Op Y</td>
<td>06 3</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>BCC</strong></td>
<td>Branch on carry clear</td>
<td>Branch on C=0</td>
<td>Relative BCC Op A</td>
<td>90 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>BCS</strong></td>
<td>Branch on carry set</td>
<td>Branch on C=1</td>
<td>Relative BCS Op A</td>
<td>80 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>BEQ</strong></td>
<td>Branch on result zero</td>
<td>Branch on Z=1</td>
<td>Relative BEQ Op A</td>
<td>80 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>BIT</strong></td>
<td>Test bits in memory with accumulator</td>
<td>A.M, M2 — N, M1 — Y</td>
<td>Zero Page BIT# Op A</td>
<td>2A 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute BIT# Op X</td>
<td>25 3</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(direct.X) BIT# Op X</td>
<td>37 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>BMI</strong></td>
<td>Branch on result minus</td>
<td>Branch on N=1</td>
<td>Relative BMI Op A</td>
<td>30 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>BNE</strong></td>
<td>Branch on result not zero</td>
<td>Branch on Z=0</td>
<td>Relative BNE Op A</td>
<td>90 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>BPL</strong></td>
<td>Branch on result plus</td>
<td>Branch on N=0</td>
<td>Relative BPL Op A</td>
<td>10 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>BRK</strong></td>
<td>Force Break</td>
<td>Implied</td>
<td>BRK*</td>
<td>00 1</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>BVC</strong></td>
<td>Branch on overflow clear</td>
<td>Branch on V=0</td>
<td>Relative BVC Op A</td>
<td>90 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX OP Code</th>
<th>No. Bytes</th>
<th>'P' Status Reg.</th>
<th>N Z C I O V</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BVS</strong></td>
<td>Branch on overflow set</td>
<td>Branch on V=1</td>
<td>Relative BVS Op A</td>
<td>70 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>CLC</strong></td>
<td>Clear carry flag</td>
<td>0 — C</td>
<td>Implied</td>
<td>CLC</td>
<td>18 1</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>CLD</strong></td>
<td>Clear decimal mode</td>
<td>0 — D</td>
<td>Implied</td>
<td>CLD</td>
<td>08 1</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>CLI</strong></td>
<td>Clear interrupt</td>
<td>0 — I</td>
<td>Implied</td>
<td>CLI</td>
<td>66 1</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>CLV</strong></td>
<td>Clear overflow flag</td>
<td>0 — Y</td>
<td>Implied</td>
<td>CLV</td>
<td>80 1</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>CMP</strong></td>
<td>Compare memory and accumulator</td>
<td>A — M</td>
<td>Immediate CMP #Op A</td>
<td>6A 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page CMP Op A</td>
<td>50 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute CMP Op X</td>
<td>46 3</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, X CMP Op Y</td>
<td>3A 4</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, Y (direct.X) CMP Op X</td>
<td>28 5</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, Y (indirect.Y) CMP Op Y</td>
<td>26 5</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>CPX</strong></td>
<td>Compare memory and index X</td>
<td>X — M</td>
<td>Immediate CPX #Op A</td>
<td>80 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page CPX Op A</td>
<td>70 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute CPX Op X</td>
<td>66 3</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, X CPX Op Y</td>
<td>5A 4</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, Y (direct.X) CPX Op X</td>
<td>50 5</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, Y (indirect.Y) CPX Op Y</td>
<td>48 5</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>CPY</strong></td>
<td>Compare memory and index Y</td>
<td>Y — M</td>
<td>Immediate CPY #Op A</td>
<td>80 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page CPY Op A</td>
<td>70 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute CPY Op X</td>
<td>66 3</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, X CPY Op Y</td>
<td>5A 4</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, Y (direct.X) CPY Op X</td>
<td>50 5</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, Y (indirect.Y) CPY Op Y</td>
<td>48 5</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>DEC</strong></td>
<td>Decrement memory by one</td>
<td>M — 1</td>
<td>M-1</td>
<td>DEC Op A</td>
<td>6B 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page DEC Op A</td>
<td>5B 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute DEC Op X</td>
<td>4D 3</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, X DEC Op Y</td>
<td>3B 4</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, Y (direct.X) DEC Op X</td>
<td>3D 5</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, Y (indirect.Y) DEC Op Y</td>
<td>3B 5</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>DEX</strong></td>
<td>Decrement index X by one</td>
<td>X — 1</td>
<td>X-1</td>
<td>DEX Op A</td>
<td>0B 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page DEX Op A</td>
<td>5B 2</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute DEX Op X</td>
<td>4D 3</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, X DEX Op Y</td>
<td>3B 4</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, Y (direct.X) DEX Op X</td>
<td>3D 5</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute, Y (indirect.Y) DEX Op Y</td>
<td>3B 5</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>DEY</strong></td>
<td>Decrement index Y by one</td>
<td>Y — 1</td>
<td>Y-1</td>
<td>DEY Op A</td>
<td>88 1</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
</tbody>
</table>
# INSTRUCTION CODES

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX OP Code</th>
<th>No. Bytes</th>
<th>&quot;P&quot; Status Reg</th>
<th>N Z C I D B Y</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EOR</strong> &quot;Exclusive-Or&quot; memory with accumulator</td>
<td>A VM A</td>
<td>Immediate</td>
<td>EOR Oper</td>
<td>46</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>EOR Oper</td>
<td>45</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>EOR Oper X</td>
<td>55</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>EOR Oper</td>
<td>40</td>
<td>3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>EOR Oper X</td>
<td>50</td>
<td>3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>EOR Oper Y</td>
<td>50</td>
<td>3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect) X</td>
<td>EOR Oper X</td>
<td>41</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect) Y</td>
<td>EOR Oper Y</td>
<td>51</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>INC</strong> Increment memory by one</td>
<td>M + 1 M</td>
<td>Zero Page</td>
<td>INC Oper</td>
<td>66</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>INC Oper X</td>
<td>66</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>INC Oper</td>
<td>88</td>
<td>3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>INC Oper X</td>
<td>88</td>
<td>3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>INX</strong> Increment index X by one</td>
<td>X + 1 X</td>
<td>Implied</td>
<td>INX</td>
<td>66</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>INY</strong> Increment index Y by one</td>
<td>Y + 1 Y</td>
<td>Implied</td>
<td>INY</td>
<td>66</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>JMP</strong> Jump to new location</td>
<td>(PC-1) PCL</td>
<td>Absolute</td>
<td>JMP Oper</td>
<td>4C</td>
<td>3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(PC-2) PCH</td>
<td>JMP Oper</td>
<td>4C</td>
<td>3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>JSR</strong> Jump to new location saving return address</td>
<td>PC-2+ (PC-1) PCL</td>
<td>Absolute</td>
<td>JSR Oper</td>
<td>20</td>
<td>3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>LDA</strong> Load accumulator with memory</td>
<td>M A</td>
<td>Immediate</td>
<td>LDA Oper</td>
<td>AB</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LDA Oper</td>
<td>AB</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>LDA Oper X</td>
<td>BB</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>LDA Oper</td>
<td>BB</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>LDA Oper X</td>
<td>BB</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>LDA Oper Y</td>
<td>BB</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect) X</td>
<td>LDA Oper X</td>
<td>BB</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect) Y</td>
<td>LDA Oper Y</td>
<td>BB</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>LDX</strong> Load index X with memory</td>
<td>M X</td>
<td>Immediate</td>
<td>LDX Oper</td>
<td>A2</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LDX Oper</td>
<td>A2</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>LDX Oper X</td>
<td>B2</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>LDX Oper</td>
<td>A2</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>LDX Oper X</td>
<td>B2</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>LDY</strong> Load index Y with memory</td>
<td>M Y</td>
<td>Immediate</td>
<td>LDY Oper</td>
<td>A2</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LDY Oper</td>
<td>A2</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>LDY Oper X</td>
<td>B2</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>LDY Oper</td>
<td>A2</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>LDY Oper X</td>
<td>B2</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX OP Code</th>
<th>No. Bytes</th>
<th>&quot;P&quot; Status Reg</th>
<th>N Z C I D B Y</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LSR</strong> Shift right one bit (memory or accumulator)</td>
<td>(See Figure 1)</td>
<td>Accumulator</td>
<td>LSR Oper</td>
<td>4A</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LSR Oper</td>
<td>45</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>LSR Oper X</td>
<td>55</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>LSR Oper</td>
<td>45</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>LSR Oper X</td>
<td>55</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>NOP</strong> No operation</td>
<td>No Operation</td>
<td>Implied</td>
<td>NOP</td>
<td>EA</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>ORA</strong> &quot;OR&quot; memory with accumulator</td>
<td>A VM A</td>
<td>Immediate</td>
<td>ORA Oper</td>
<td>09</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>ORA Oper</td>
<td>09</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>ORA Oper X</td>
<td>09</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>ORA Oper</td>
<td>09</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>ORA Oper X</td>
<td>09</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>ORA Oper Y</td>
<td>10</td>
<td>3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect) X</td>
<td>ORA Oper X</td>
<td>10</td>
<td>3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect) Y</td>
<td>ORA Oper Y</td>
<td>11</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>PHA</strong> Push accumulator on stack</td>
<td>A</td>
<td>Implied</td>
<td>PHA</td>
<td>45</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>PHP</strong> Push processor status on stack</td>
<td>P</td>
<td>Implied</td>
<td>PHP</td>
<td>09</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>PLA</strong> Pull accumulator from stack</td>
<td>A</td>
<td>Implied</td>
<td>PLA</td>
<td>66</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>PLP</strong> Pull processor status from stack</td>
<td>P</td>
<td>Implied</td>
<td>PLP</td>
<td>28</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>ROL</strong> Rotate one bit left (memory or accumulator)</td>
<td>(See Figure 2)</td>
<td>Accumulator</td>
<td>ROL Oper</td>
<td>2A</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>ROL Oper</td>
<td>26</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>ROL Oper X</td>
<td>36</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>ROL Oper</td>
<td>26</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>ROL Oper X</td>
<td>36</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>ROR</strong> Rotate one bit right (memory or accumulator)</td>
<td>(See Figure 3)</td>
<td>Accumulator</td>
<td>ROR Oper</td>
<td>6A</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>ROR Oper</td>
<td>66</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>ROR Oper X</td>
<td>76</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>ROR Oper</td>
<td>66</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>ROR Oper X</td>
<td>76</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
## INSTRUCTION CODES

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX Op Code</th>
<th>No. Bytes</th>
<th>P' Status Reg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTI</td>
<td>Return from interrupt</td>
<td>FI PC</td>
<td>Implied</td>
<td>RTI</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>RTS</td>
<td>Return from subroutine</td>
<td>FI, PC-1 → PC</td>
<td>Implied</td>
<td>RTS</td>
<td>1</td>
<td>--</td>
</tr>
<tr>
<td>SBC</td>
<td>Subtract memory from accumulator with borrow</td>
<td>A - M • C → A</td>
<td>Immediate</td>
<td>SBC, Oper</td>
<td>ED</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>SBC Oper</td>
<td>E5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>SBC Oper</td>
<td>F8</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>SBC Oper</td>
<td>ED</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>SBC Oper</td>
<td>F8</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect X)</td>
<td>SBC (Oper) X</td>
<td>E1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect Y)</td>
<td>SBC (Oper) Y</td>
<td>F1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>SEC</td>
<td>Set carry flag</td>
<td>1 → C</td>
<td>Implied</td>
<td>SEC</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>SED</td>
<td>Set decimal mode</td>
<td>1 → D</td>
<td>Implied</td>
<td>SED</td>
<td>F8</td>
<td>1</td>
</tr>
<tr>
<td>SEI</td>
<td>Set interrupt disable status</td>
<td>1 → I</td>
<td>Implied</td>
<td>SEI</td>
<td>78</td>
<td>1</td>
</tr>
<tr>
<td>STA</td>
<td>Store accumulator in memory</td>
<td>A → M</td>
<td>Zero Page</td>
<td>STA Oper</td>
<td>85</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>STA Oper</td>
<td>86</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>STA Oper</td>
<td>85</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>STA Oper</td>
<td>86</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>STA Oper</td>
<td>85</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect X)</td>
<td>STA Oper</td>
<td>86</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect Y)</td>
<td>STA Oper</td>
<td>85</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>STX</td>
<td>Store index X in memory</td>
<td>X → M</td>
<td>Zero Page</td>
<td>STX Oper</td>
<td>86</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page, Y</td>
<td>STX Oper</td>
<td>86</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>STX Oper</td>
<td>86</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>STY</td>
<td>Store index Y in memory</td>
<td>Y → M</td>
<td>Zero Page, X</td>
<td>STY Oper</td>
<td>94</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page, Y</td>
<td>STY Oper</td>
<td>94</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>STY Oper</td>
<td>94</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>TAX</td>
<td>Transfer accumulator to index X</td>
<td>A → X</td>
<td>Implied</td>
<td>TAX</td>
<td>AA</td>
<td>1</td>
</tr>
<tr>
<td>TAY</td>
<td>Transfer accumulator to index Y</td>
<td>A → Y</td>
<td>Implied</td>
<td>TAY</td>
<td>AB</td>
<td>1</td>
</tr>
<tr>
<td>TSX</td>
<td>Transfer stack pointer to index X</td>
<td>S → X</td>
<td>Implied</td>
<td>TSX</td>
<td>BA</td>
<td>1</td>
</tr>
</tbody>
</table>

### Additional Operations

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX Op Code</th>
<th>No. Bytes</th>
<th>P' Status Reg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TXA</td>
<td>Transfer index X to accumulator</td>
<td>X → A</td>
<td>Implied</td>
<td>TXA</td>
<td>8A</td>
<td>1</td>
</tr>
<tr>
<td>TXS</td>
<td>Transfer index X to stack pointer</td>
<td>X → S</td>
<td>Implied</td>
<td>TXS</td>
<td>9A</td>
<td>1</td>
</tr>
<tr>
<td>TYA</td>
<td>Transfer index Y to accumulator</td>
<td>Y → A</td>
<td>Implied</td>
<td>TYA</td>
<td>98</td>
<td>1</td>
</tr>
<tr>
<td>Hex Code</td>
<td>Operation Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>BRK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>ORA (Indirect, XI)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>NOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>ORI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>ORA — Zero Page</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>ASL — Zero Page</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>ORA — Immediate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0A</td>
<td>ASL — Accumulator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0B</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0C</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0D</td>
<td>ORA — Absolute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0E</td>
<td>ASL — Absolute, X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0F</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>BPL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>ORA (Indirect, Y)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>ORA — Zero Page, X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>ASL — Zero Page, X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>CLC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>ORA — Absolute, Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1A</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1C</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1D</td>
<td>ORA — Absolute, X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E</td>
<td>ASL — Absolute, X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1F</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>JSR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>AND (Indirect, X)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>BIT — Zero Page</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>AND — Zero Page</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>ROL — Zero Page</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>PLP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>AND — Immediate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2A</td>
<td>ROL — Accumulator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2B</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2C</td>
<td>BIT — Absolute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2D</td>
<td>AND — Absolute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2E</td>
<td>ROL — Absolute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**HEX OPERATION CODES**
APPLE II HARDWARE

1. Getting Started with Your APPLE II Board
2. APPLE II Switching Power Supply
3. Interfacing with the Home TV
4. Simple Serial Output
5. Interfacing the APPLE —
   Signals, Loading, Pin Connections
6. Memory —
   Options, Expansion, Map, Address
7. System Timing
8. Schematics
GETTING STARTED WITH YOUR APPLE II BOARD

INTRODUCTION

ITEMS YOU WILL NEED:

Your APPLE II board comes completely assembled and thoroughly tested. You should have received the following:

a. 1 ea. APPLE II P.C. Board complete with specified RAM memory.

b. 1 ea. d.c. power connector with cable.

c. 1 ea. 2" speaker with cable.

d. 1 ea. Preliminary Manual

e. 1 ea. Demonstration cassette tapes. (For 4K: 1 cassette (2 programs); 16K or greater: 3 cassettes.

f. 2 ea. 16 pin headers plugged into locations A7 and J14

In addition you will need:

g. A color TV set (or B & W) equipped with a direct video input connector for best performance or a commercially available RF modulator such as a “Pixi-verter”\textsuperscript{tm}

Higher channel (7-13) modulators generally provide better system performance than lower channel modulators (2-6).

h. The following power supplies (NOTE: current ratings do not include any capacity for peripheral boards.):

1. +12 Volts with the following current capacity!

   a. For 4K or 16K systems - 350mA.

   b. For 8K, 20K or 32K - 550mA.

   c. For 12K, 24K, 36K or 48K - 850mA.

2. +5 Volts at 1.6 amps

3. -5 Volts at WmA.

4. OPTIONAL: If -12 Volts is required by your keyboard. (If using an APPLE II supplied keyboard, you will need -12V at 50mA.)
i. An audio cassette recorder such as a Panasonic model RQ-309 DS which is used to load and save programs.

An ASCII encoded keyboard equipped with a "reset" switch.

k. Cable for the following:

1. Keyboard to APPLE II P.C.B.
2. Video out 75 ohm cable to TV or modulator
3. Cassette to APPLE II P.C.B. (1 or 2)

Optionally you may desire:

1. Game paddles or pots with cables to APPLE II Game I/O connector. (Several demo programs use PDL(0) and "Pong" also uses PDL(1)).

m. Case to hold all the above

Final Assembly Steps

1. Using detailed information on pin functions in hardware section of manual, connect power supplies to d.c. cable assembly. Use both ground wires to minimize resistance. With cable assembly disconnected from APPLE II motherboard, turn on power supplies and verify voltages on connector pins. Improper supply connections such as reverse polarity can severely damage your APPLE II.

2. Connect keyboard to APPLE II by unplugging leader in location A7 and wiring keyboard cable to it, then plug back into APPLE II P.C.B.

3. Plug in speaker cable.

4. Optionally connect one or two game paddles using leader supplied in socket located at J14.

5. Connect video cable.

6. Connect cable from cassette monitor output to APPLE II cassette input.

7. Check to see that APPLE II board is not contacting any conducting surface.

8. With power supplies turned off, plug in power connector to mother board then recheck all cableing.
POWER UP

1. Turn power on. If power supplies overload, immediately turn off and recheck power cable wiring. Verify operating supply voltages are within +3% of nominal value.

2. You should now have random video display. If not check video level pot on mother board, full clockwise is maximum video output. Also check video cables for opens and shorts. Check modulator if you are using one.

3. Press reset button. Speaker should beep and a "*" prompt character with a blinking cursor should appear in lower left on screen.

4. Press "esc" button, release and type a "(0" (shift-P) to clear screen. You may now try "Monitor" commands if you wish. See details in "Ionitor" software section.

RUNNING BASIC

1. Turn power on; press reset button; type "control B" and press return button. A ">" prompt character should appear on screen indicating that you are now in BASIC.

2. Load one of the supplied demonstration cassettes into recorder. Set recorder level to approximately 5 and start recorder. Type "LOAD" and return. First beep indicates that APPLE II has found beginning of program; second indicates end of program followed by ">" character on screen. If error occurs on loading, try a different demo tape or try changing cassette volume level.

3. Type RUN and carriage return to execute demonstration program. Listings of these are included in the last section of this manual.
Switching power supplies generally have both advantages and peculiarities not generally found in conventional power supplies. The Apple II user is urged to review this section.

Your Apple II is equipped with an AC line voltage filter and a three wire AC line cord. It is important to make sure that the third wire is returned to earth ground. Use a continuity checker or ohmmeter to ensure that the third wire is actually returned to earth. Continuity should be checked for between the power supply case and an available water pipe for example. The line filter, which is of a type approved by domestic (U.L. CSA) and international (VDE) agencies must be returned to earth to function properly and to avoid potential shock hazards.

The APPLE II power supply is of the "flyback" switching type. In this system, the AC line is rectified directly, "chopped up" by a high frequency oscillator and coupled through a small transformer to the diodes, filters, etc., and results in four low voltage DC supplies to run APPLE II. The transformer isolates the DC supplies from the line and is provided with several shields to prevent "hash" from being coupled into the logic or peripherals. In the "flyback" system, the energy transferred through from the AC line side to DC supply side is stored in the transformer's inductance on one-half of the operating cycle, then transferred to the output filter capacitors on the second half of the operating cycle. Similar systems are used in TV sets to provide horizontal deflection and the high voltages to run the CRT.

Regulation of the DC voltages is accomplished by controlling the frequency at which the converter operates; the greater the output power needed, the lower the frequency of the converter. If the converter is overloaded, the operating frequency will drop into the audible range with squeels and squawks warning the user that something is wrong.

All DC outputs are regulated at the same time and one of the four outputs (the +5 volt supply) is compared to a reference voltage with the difference error fed to a feedback loop to assist the oscillator in running at the needed frequency. Since all DC outputs are regulated together, their voltages will reflect to some extent unequal loadings.
For example; if the +5 supply is loaded very heavily, then all other supply voltages will increase in voltage slightly; conversely, very light loading on the +5 supply and heavy loading on the +12 supply will cause both it and the others to sag lightly. If precision reference voltages are needed for peripheral applications, they should be provided for in the peripheral design.

In general, the APPLE II design is conservative with respect to component ratings and operating temperatures. An over-voltage crowbar shutdown system and an auxiliary control feedback loop are provided to ensure that even very unlikely failure modes will not cause damage to the APPLE II computer system. The over-voltage protection references to the DC output voltages only. The AC line voltage input must be within the specified limits, i.e., 107V to 132V.

Under no circumstances, should more than 140 VAC be applied to the input of the power supply. Permanent damage will result.

Since the output voltages are controlled by changing the operating frequency of the converter, and since that frequency has an upper limit determined by the switching speed of power transistors, there then must be a minimum load on the supply; the Apple II board with minimum memory (4K) is well above that minimum load. However, with the board disconnected, there is no load on the supply, and the internal over-voltage protection circuitry causes the supply to turn off. A 9 watt load distributed roughly 50-50 between the +5 and +12 supply is the nominal minimum load.

Nominal load current ratios are: The +12V supply load is ½ that of the +5V. The -5V supply load is 1/10 that of the +5V. The -12V supply load is 1/10, that of the +5V.

The supply voltages are +5.0 ± 0.15 volts, +11.8 ± 0.5 volts, -12.0 ± 1V, -5.2 ± 0.5 volts. The tolerances are greatly reduced when the loads are close to nominal.

The Apple II power supply will power the Apple II board and all present and forthcoming plug-in cards, we recommend the use of low power TTL, CMOS, etc. so that the total power drawn is within the thermal limits of the entire system. In particular, the user should keep the total power drawn by any one card to less than 1.5 watts, and the total current drawn by all the cards together within the following limits:

+12V - use no more than 250 mA
+5V - use no more than 500 mA
-5V - use no more than 200 mA
-12V - use no more than 200 mA

The power supply is allowed to run indefinetly under short circuit or open circuit conditions.

CAUTION: There are dangerous high voltages inside the power supply case. Much of the internal circuitry is NOT isolated from the power line, and special equipment is needed for service. NO REPAIR BY THE USER IS ALLOWED.
Accessories are available to aid the user in connecting the Apple II system to a home color TV with a minimum of trouble. These units are called "RF Modulators" and they generate a radio frequency signal corresponding to the carrier of one or two of the lower VHF television bands: 61.25 MHz (channel 3) or 67.25 MHz (channel 4). This RF signal is then modulated with the composite video signal generated by the Apple II.

Users report success with the following RF modulators:

the "PixieVerter" (a kit)
ATV Research
13th and Broadway
Dakota City, Nebraska 68731

the "TV-1" (a kit)
UHF Associates
6037 Haviland Ave.
Whittier, CA 90601

the "Sup-r-Mod" by (assembled & tested)
M&R Enterprises
P.O. Box 1011
Sunnyvale, CA 94088

the RF Modulator (a P.C. board)
Electronics Systems
P.O. Box 212
Burlingame, CA 94010

Most of the above are available through local computer stores.

The Apple II owner who wishes to use one of these RF Modulators should read the following notes carefully.

All these modulators have a free running transistor oscillator. The M&R Enterprises unit is pre-tuned to Channel 4. The PixieVerter and the TV-1 have tuning by means of a jumper on the P.C. board and a small trimmer capacitor. All these units have a residual FM which may cause trouble if the TV set in use has a IF pass band with excessive ripple. The unit from M&R has the least residual FM.

All the units except the M&R unit are kits to be built and tuned by the customer. All the kits are incomplete to some extent. The unit from Electronics Systems is just a printed circuit board with assembly instructions. The kits from UHF Associates and ATV do not have an RF cable or a shielded box or a balun transformer, or an antenna switch. The M&R unit is complete.

Some cautions are in order. The Apple II, by virtue of its color graphics capability, operates the TV set in a linear mode rather than the 100% contrast mode satisfactory for displaying text. For this reason, radio frequency interference (RFI) generated by a computer (or peripherals) will beat with the
carrier of the RF modulator to produce faint spurious background patterns (called "worms"). This RFI "trash" must be of quite a low level if worms are to be prevented. In fact, these spurious beats must be 40 to 50dB below the signal level to reduce worms to an acceptable level. When it is remembered that only 2 to 6 mV (across 300Ω) is presented to the VHF input of the TV set, then stray RFI getting into the TV must be less than 500μV to obtain a clean picture. Therefore we recommend that a good, co-ax cable be used to carry the signal from any modulator to the TV set, such as RG/59u (with copper shield), Belden #8241 or an equivalent miniature type such as Belden #8218. We also recommend that the RF modulator be closed in a tight metal box (an unpainted die cast aluminum box such as Pomona #2428). Even with these precautions, some trouble may be encountered with worms, and can be greatly helped by threading the coax cable connecting the modulator to the TV set repeatedly through a Ferrite toroid core. Apple Computer supplies these cores in a kit: along with a 4 circuit connector/cable assembly to match the auxiliary video connector found on the Apple II board. This kit has order number A2M010X. The M&R "Sup-r-Mod is supplied with a coax cable and toroids.

Any computer containing fast switching logic and high frequency clocks will radiate some "radio frequency energy. Apple II is equipped with a good line filter and many other precautions have been taken to minimize radiated energy. The user is urged not to connect "antennas" to this computer; wires strung about carrying clocks and data will act as antennas, and subsequent radiated energy may prove to be a nuisance.

Another caution concerns possible long term effects on the TV picture tube. Most home TV sets have "Brightness" and "Contrast" controls with a very wide range of adjustment. When an un-changing picture is displayed with high brightness for a long period, a faint discoloration of the TV CRT may occur as an inverse pattern observable with the TV set turned off. This condition may be avoided by keeping the "Brightness "turned down slightly and "Contrast" moderate.
The Apple II is equipped with a 16 pin DIP socket most frequently used to connect potentiometers, switches, etc. to the computer for paddle control and other game applications. This socket, located at J-14, has outputs available as well. With an appropriate machine language program, these output lines may be used to serialize data in a format suitable for a teletype. A suitable interface circuit must be built since the outputs are merely LSTTL and won't run a teletype without help. Several interface circuits are discussed below and the user may pick the one best suited to his needs.

The ASR - 33 Teletype

The ASR - 33 Teletype of recent vintage has a transistor circuit to drive its solenoids. This circuit is quite easy to interface to, since it is provided with its own power supply. (Figure 1a) It can be set up for a 20mA current loop and interfaced as follows (whether or not the teletype is strapped for full duplex or half duplex operation):

a) The yellow wire and purple wire should both go to terminal 9 of Terminal Strip X. If the purple wire is going to terminal 8, then remove it and relocate it at terminal 9. This is necessary to change from the 60mA current loop to the 20mA current loop.

b) Above Terminal Strip X is a connector socket identified as "2". Pin 8 is the input line + or high; Pin 7 is the input line - or low. This connector mates with a Molex receptacle model 1375 #03-09-2151 or #03-09-2153. Recommended terminals are Molex #02-09-2136. An alternate connection method is via spade lugs to Terminal Strip X, terminal 7 (the + input line) and 6 (the - input line).

c) The following circuit can be built on a 16 pin DIP component carrier and then plugged into the Apple's 16 pin socket found at J-14: (The junction of the 3.3k resistor and the transistor base lead is floating). Pins 16 and 9 are used as tie points as they are unconnected on the Apple board. (Figure 1a).
The "RS - 232 Interface"

For this interface to be legitimate, it is necessary to twice invert the signal appearing at J-14 pin 15 and have it swing more than 5 volts both above and below ground. The following circuit does that but requires that both +12 and -12 supplies be used. (Figure 2) Snipping off pins on the DIP-component carrier will allow the spare terminals to be used for tie points. The output ground connects to pin 7 of the DB-25 connector. The signal output connects to pin 3 of the DB-25 connector. The "protective" ground wire normally found on pin 1 of the DB-25 connector may be connected to the Apple's base plate if desired. Placing a #4 lug under one of the four power supply mounting screws is perhaps the simplest method. The +12 volt supply is easily found on the auxiliary Video connector (see Figure S-11 or Figure 7 of the manual). The -12 volt supply may be found at pin 33 of the peripheral connectors (see Figure 4) or at the power supply connector (see Figure 5 of the manual).

A Serial Out Machine Center Language Program

Once the appropriate circuit has been selected and constructed a machine language program is needed to drive the circuit. Figure 3 lists such a teletype output machine language routine. It can be used in conjunction with an Integer BASIC program that doesn't require page $3000 hex of memory. This program resides in memory from $3700 to $3E9. Columns three and four of the listing show the op-code used. To enter this program into the Apple II the following procedure is followed:

Entering Machine Language Program

1. Power up Apple II
2. Depress and release the "RESET" key. An asterick and flashing cursor should appear on the left hand side of the screen below the random text matrix.
3. Now type in the data from columns one, two and three for each line from $3700 to $3E9. For example, type in "3700: A9 82" and then depress and release the "RETURN" key. Then repeat this procedure for the data at $3720 and on until you complete entering the program.

Executing this Program

1. From BASIC a CALL 8800 ($3700) will start the execution of this program. It will use the teletype or suitable 80 column printer as the primary output device.
2. PR#Ø will inactivate the printer transferring control back to the Video monitor as the primary output device.

3. In Monitor mode $3700 activates the printer and hitting the "RESET" key exits the program.

Saving the Machine Language Program

After the machine language program has been entered and checked for accuracy it should, for convenience, be saved on tape - that is unless you prefer to enter it by keyboard every time you want to use it.

The way it is saved is as follows:
1. Insert a blank program cassette into the tape recorder and rewind it.

2. Hit the "RESET" key. The system should move into Monitor mode. An asterisk "*" and flashing cursor should appear on the left-hand side of the screen.

3. Type in "370.03E9W 370.03E9W".

4. Start the tape recorder in record mode and depress the "RETURN" key.

5. When the program has been written to tape, the asterick and flashing cursor will reappear.

The Program

After entering, checking and saving the program perform the following procedure to get a feeling of how the program is used:
1. BC (control B) into BASIC

2. Turn the teletype (printer on)

3. Type in the following
   10 CALL 880
   15 PRINT "ABCD...XYZ0123456789"
   20 PR#Ø
   25 END

4. Type in RUN and hit the "RETURN" key. The text in line 15 should be printed on the teletype and control is returned to the keyboard and Video monitor.
Line 10 activates the teletype machine routine and all "PRINT" statements following it will be printed to the teletype until a PR#Ø statement is encountered. Then the text in line 15 will appear on the teletype's output. Line 20 deactivates the printer and the program ends on line 25.

Conclusion

With the circuits and machine language program described in this paper the user may develop a relatively simple serial output interface to an ASR-3 or RS-232 compatible printers. This circuit can be activated through BASIC or monitor modes. And is a valuable addition to any users program library.
(a) FIGURE 1 ASR-33

(b) FIGURE 2 RS-232
FIGURE 3a

****WARNING: OPERAND OVERFLOW IN LINE 27

***WARNING: OPERAND OVERFLOW IN LINE 27

** TECHTEFEL Routines**

**TITLE TELETYPE DRIVER ROUTINES’

1  " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " 

**TABLE TELETYPE DRIVER ROUTINES’**

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0370</td>
<td>A9 82</td>
<td>0370</td>
<td>LDA #TTOUT</td>
</tr>
<tr>
<td>0372</td>
<td>85 36</td>
<td>0372</td>
<td>STA CSWL</td>
</tr>
<tr>
<td>0374</td>
<td>A9 03</td>
<td>0374</td>
<td>LDA #TTOUT/256</td>
</tr>
<tr>
<td>0376</td>
<td>85 37</td>
<td>0376</td>
<td>STA CSWL+1</td>
</tr>
<tr>
<td>0378</td>
<td>A9 48</td>
<td>0378</td>
<td>LDA #72</td>
</tr>
<tr>
<td>037A</td>
<td>85 21</td>
<td>037A</td>
<td>STA WNDWIDTH</td>
</tr>
<tr>
<td>037C</td>
<td>A5 24</td>
<td>037C</td>
<td>LDA CH</td>
</tr>
<tr>
<td>037E</td>
<td>8D F8</td>
<td>037E</td>
<td>STA COLCNT</td>
</tr>
<tr>
<td>0381</td>
<td>60 35</td>
<td>0381</td>
<td>RTS</td>
</tr>
<tr>
<td>0382</td>
<td>48 36</td>
<td>0382</td>
<td>TTOUT: PHA</td>
</tr>
<tr>
<td>0383</td>
<td>48 37</td>
<td>0383</td>
<td>PHA</td>
</tr>
<tr>
<td>0384</td>
<td>AD F8</td>
<td>0384</td>
<td>TTOUT2: LDA COLCNT</td>
</tr>
<tr>
<td>0387</td>
<td>C5 24</td>
<td>0387</td>
<td>CMP CH</td>
</tr>
<tr>
<td>0389</td>
<td>68 40</td>
<td>0389</td>
<td>PLA</td>
</tr>
<tr>
<td>038A</td>
<td>B0 03</td>
<td>038A</td>
<td>BCS TESTCTRL</td>
</tr>
<tr>
<td>038C</td>
<td>48 42</td>
<td>038C</td>
<td>PHA</td>
</tr>
<tr>
<td>038D</td>
<td>A9 AO</td>
<td>038D</td>
<td>LDA #$A0</td>
</tr>
<tr>
<td>038F</td>
<td>2C CO</td>
<td>038F</td>
<td>TESTCTRL BIT RTS1</td>
</tr>
<tr>
<td>0392</td>
<td>F0 03</td>
<td>0392</td>
<td>BEQ PRNTIT</td>
</tr>
<tr>
<td>0394</td>
<td>EE F8</td>
<td>0394</td>
<td>INC COLCNT</td>
</tr>
<tr>
<td>0397</td>
<td>20 C1</td>
<td>0397</td>
<td>PRNTIT: JSR DOCHAR</td>
</tr>
<tr>
<td>039A</td>
<td>68 48</td>
<td>039A</td>
<td>PLA</td>
</tr>
<tr>
<td>039B</td>
<td>48 49</td>
<td>039B</td>
<td>PHA TTOUT2</td>
</tr>
<tr>
<td>039C</td>
<td>90 E6</td>
<td>039C</td>
<td>BCC #$0D</td>
</tr>
<tr>
<td>039E</td>
<td>49 OD</td>
<td>039E</td>
<td>FOR A</td>
</tr>
<tr>
<td>03A0</td>
<td>0A 52</td>
<td>03A0</td>
<td>ASL FINISH</td>
</tr>
<tr>
<td>03A1</td>
<td>DO OD</td>
<td>03A1</td>
<td>BNE</td>
</tr>
</tbody>
</table>

---

**FIGURE 3a**

---

**FIGURE 3a**
TELETYPİE DRIVER ROUTİNES

3:42 P.M., 11/13/1977

03A3: 8D F8 07 54 STA COLCNT ; CLEAR COLUMN COUNT
03A6: A9 8A 55 LDA #38A ; NOW DO LINE FEED
03A8: 20 C1 03 56 JSR DOCHAR
03AB: A9 58 57 LDA #153
03AD: 20 A8 FC 58 JSR 7AIT ; 200MSEC DELAY FOR LIB
0330: AD F8 07 59 LDA COLCNT ; CHECK IF IN MARGIN
0333: F0 08 60 3E0 SETCH ; FOR CR, RESET CH
0335: E5 21 61 S3C #7DTH
0337: E9 F7 62 BCC RETURN
0339: 90 04 63 ADC #11F ; ADJUST CH
033D: 85 24 65 STA CH
033F: 68 66 SETCH: PLA
0340: 60 67 RETURN: RTS
03C0: 6C 68 RTS1: RTS
03C1: 8F 77 ; RETURN TO CALLER
03C3: 8C 78 07 69 A CHARACTER ROUTINE:
03C4: 88 08 70 DOCHAR: PHP YSAVE
03C5: 80 08 71 ; SAVE STATUS.
03C7: A0 08 71 CLC #SOS ; 11 BITS (1 START, 1 2
03C9: 18 72 PHA ; BEGIN 7TH SPACE (ST2
03C3: 48 1F 73 TTOUT3: LDA MARKOUT ; SEND A SPACE
03C3: 80 05 74 3CC SPACE
03CE: AD 59 C0 75 LDA TTOUT4
0300: 90 03 76 ; SEND A MARK
0303: AD 58 C0 77 MARKOUT:
0305: A9 D7 78 LDA MARK
0306: 48 79 TTOUT4: PHA #4D7 ; DELAY 9.091 MSEC FOR
0308: A9 20 80 LDA
0309: 4A 81 DLY1: LSR #$20
03D8: A9 20 80 BCC A
03D9: 4A 81 DLY2: PLA DLY2
03DC: 68 83 SBC
03E0: 6A 84 3NE #101
03E1: D0 E3 86 PLP YSAVE ; RESTORE Y-REG.
03E2: AC 78 07 87 RTS ; RESTORE STATUS
03E3: 28 88 ; RETURN
03E5: 60 89
03E8: 90
03E9: 91

********SUCCESSFUL ASSEMBLY: NO ERRORS

FIGURE 3b
CROSS-REFERENCE: TELETYPE DRIVER ROUTINES

CH 0024 0033 0039 0065
COLCNT 0718 0034 0038 0046 0054 0059
05YL 0036 0028 0030
DLYI 0305 0085
DLY2 0308 0082
DOCHAR 0301 0047 0056
FINISH 0330 0053
MARK C058 0077
MARKOUT 0300 0074
PRNTIT 0397 0045
RETURN 038F 0063
RTS1 0300 0044
SETCH 0330 0060
SPACE C059 0075
TESTCTRL 033F 0041
TTINIT 0370
TTOUT 0332 0027 0029
TTOUT2 0384 0050
TTOUT3 03C8 0089
TTOUT4 0303 0076
WAIT FCAB 0058
WNDWTH 0021 0032 0061
YSAVE 0778 0069 0090
ILE:

FIGURE 3c
INTERFACING THE APPLE

This section defines the connections by which external devices are attached to the APPLE II board. Included are pin diagrams, signal descriptions, loading constraints and other useful information.

TABLE OF CONTENTS

1. CONNECTOR LOCATION DIAGRAM
2. CASSETTE DATA JACKS (2 EACH)
3. GAME I/O CONNECTOR
4. KEYBOARD CONNECTOR
5. PERIPHERAL CONNECTORS (8 EACH)
6. POWER CONNECTOR
7. SPEAKER CONNECTOR
8. VIDEO OUTPUT JACK
9. AUXILIARY VIDEO OUTPUT CONNECTOR
CASSETTE JACKS

A convenient means for interfacing an inexpensive audio cassette tape recorder to the APPLE II is provided by these two standard (3.5mm) miniature phone jacks located at the back of the APPLE II board.

CASSETTE DATA IN JACK: Designed for connection to the "EARPHONE" or "MONITOR" output found on most audio cassette tape recorders. \( V_{IN} = 1Vpp \) (nominal), \( Z_{IN} = 12K \) Ohms. Located at K12 as illustrated in Figure

CASSETTE DATA OUT JACK: Designed for connection to the "MIC" or "MICROPHONE" input found on most audio cassette tape recorders. \( V_{OUT} = 25 \) mV into 17 Ohms, \( Z_{OUT} = 100 \) Ohms. Located at K13 as illustrated in Figure 1.

GAME I/O CONNECTOR

The Game I/O Connector provides a means for connecting paddle controls, lights and switches to the APPLE II for use in controlling video games, etc. It is a 16 pin IC socket located at J14 and is illustrated in Figure 1 and 2.
SIGNAL DESCRIPTIONS FOR GAME I/O

ANG-AN3: 8 addresses (C058-C05F) are assigned to selectively "SET" or "CLEAR" these four "ANNUNCIATOR" outputs. Envisioned to control indicator lights, each is a 74LSxx series TTL output and must be buffered if used to drive lamps.

CQ40 STB: A utility strobe output. Will go low during 9/2 of a read or write cycle to addresses C040-C04F. This is a 74LSxx series TTL output.

GND: System circuit ground. 0 Volt line from power supply.

NC: No connection.

PDL0-PDL3: Paddle control inputs. Requires a 0-150K ohm variable resistance and +5V for each paddle. Internal 100 ohm resistors are provided in series with external pot to prevent excess current if pot goes completely to zero ohms.

SW0-SW2: Switch inputs. Testable by reading from addresses C061-C063 (or C069-C06B). These are uncommitted 74LSxx series inputs.

+5V: Positive 5-Volt supply. To avoid burning out the connector pin, current drain MUST be less than 100mA.

KEYBOARD CONNECTOR

This connector provides the means for connecting as ASCII keyboard to the APPLE II board. It is a 16 pin IC socket located at A7 and is illustrated in Figures 1 and 3.

Figure 3  KEYBOARD CONNECTOR

TOP VIEW

( Front Edge of PC Board)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>16 N.C.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+5V</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>STROBE</td>
<td>15 -12V</td>
</tr>
<tr>
<td>3</td>
<td>RESET</td>
<td>14 N.C.</td>
</tr>
<tr>
<td>4</td>
<td>N.C.</td>
<td>13 B2</td>
</tr>
<tr>
<td>5</td>
<td>B6</td>
<td>12 B1</td>
</tr>
<tr>
<td>6</td>
<td>B5</td>
<td>11 B4</td>
</tr>
<tr>
<td>7</td>
<td>B7</td>
<td>10 B3</td>
</tr>
<tr>
<td>8</td>
<td>GND</td>
<td>9 N.C.</td>
</tr>
</tbody>
</table>

LOCATION A7
SIGNAL DESCRIPTION FOR KEYBOARD INTERFACE

B1-B7: 7 bit ASCII data from keyboard, positive logic (high level = "1"). TTL logic levels expected.

GND: System circuit ground. Ø Volt line from power supply.

NC: No connection.

RESET: System reset input. Requires switch closure to ground.

STROBE: Strobe output from keyboard. The APPLE II recognizes the positive going edge of the incoming strobe.

+5V: Positive 5-Volt supply. To avoid burning out the connector pin, current drain MUST be less than 100mA.

-12V: Negative 12-Volt supply. Keyboard should draw less than 50mA.

PERIPHERAL CONNECTORS

The eight Peripheral Connectors mounted near the back edge of the APPLE II board provide a convenient means of connecting expansion hardware and peripheral devices to the APPLE II I/O Bus. These are Winchester #2HW25C0-111 (or equivalent) pin card edge connectors with pins on .10" centers. Location and pin outs are illustrated in Figures 1 and 4.

SIGNAL DESCRIPTION FOR PERIPHERAL I/O

AO-A15: 16 bit system address bus. Addresses are set up by the 6502 within 300nS after the beginning of 0₁. These lines will drive up to a total of 16 standard TTL loads.

"DEVICE SELECT": Sixteen addresses are set aside for each peripheral connector. A read or write to such an address will send pin 41 on the selected connector low during 0₂ (500nS). Each will drive 4 standard TTL loads.

D0-D7: 8 bit system data bus. During a write cycle data is set up by the 6502 less than 300nS after the beginning of 0₂. During a read cycle the 6502 expects data to be ready no less than 100nS before the end of 0₂. These lines will drive up to a total of 8 total low power schottky TTL loads.
**DMA:** Direct Memory Access control output. This line has a 3K Ohm pullup to +5V and should be driven with an open collector output.

**DMA IN:** Direct Memory Access daisy chain input from higher priority peripheral devices. Will present no more than 4 standard TTL loads to the driving device.

**DMA OUT:** Direct Memory Access daisy chain output to lower priority peripheral devices. This line will drive 4 standard TTL loads.

**GND:** System circuit ground. Ø Volt line from power supply.

**INH:** Inhibit Line. When a device pulls this line low, all ROM's on board are disabled (Hex addressed D000 through FFFF). This line has a 3K Ohm pullup to +5V and should be driven with an open collector output.

**INT IN:** Interrupt daisy chain input from higher priority peripheral devices. Will present no more than 4 standard TTL loads to the driving device.

**INT OUT:** Interrupt daisy chain output to lower priority peripheral devices. This line will drive 4 standard TTL loads.

**I/O SELECT:** 256 addresses are set aside for each peripheral connector (see address map in "MEMORY" section). A read or write of such an address will send pin 1 on the selected connector low during Ø2 (500nS). This line will drive 4 standard TTL loads.

**I/O STROBE:** Pin 20 on all peripheral connectors will go low during Ø, of a read or write to any address C800-0FFF. This line will drive a total of 4 standard TTL loads.

**IRQ:** Interrupt request line to the 6502. This line has a 3K Ohm pullup to +5V and should be driven with an open collector output. It is active low.

**NC:** No connection.

**NMI:** Non Maskable Interrupt request line to the 6502. This line has a 3K Ohm pullup to +5V and should be driven with an open collector output. It is active low.

**Q3** A 1MHz (nonsymmetrical) general purpose timing signal. Will drive up to a total of 16 standard TTL loads.

**RDY:** 'Ready" line to the 6502. This line should change only during Ø1, and when low will halt the microprocessor at the next READ cycle. This line has a 3K Ohm pullup to +5V and should be driven with an open collector output.

**RES:** Reset line from "RESET" key on keyboard. Active low. Will drive 2 MOS loads per Peripheral Connector.
R/W: READ/WRITE line from 6502. When high indicates that a read cycle is in progress, and when low that a write cycle is in progress. This line will drive up to a total of 16 standard TTL loads.

USER 1: The function of this line will be described in a later document.

Ø0: Microprocessor phase V clock. Will drive up to a total of 16 standard TTL loads.

Ø1: Phase 1 clock, complement of Ø0. Will drive up to a total of 16 standard TTL loads.

7M: Seven MHz high frequency clock. Will drive up to a total of 16 standard TTL loads.

+12V: Positive 12-Volt supply.

+5V: Positive 5-Volt supply

-5V: Negative 5-Volt supply.

-12V: Negative 12-Volt supply.

POWER CONNECTOR

The four voltages required by the APPLE II are supplied via this AMP #9-35028-1.6 pin connector. See location and pin out in Figures 1 and 5.

PIN DESCRIPTION

GND: (2 pins) system circuit ground. 0 Volt line from power supply.

+12V: Positive 12-Volt line from power supply.

+5V: Positive 5-Volt line from power supply.

-5V: Negative 5-Volt line from power supply.

-12V: Negative 5-Volt line from power supply.
Figure 4  PERIPHERAL CONNECTORS  
(EIGHT OF EACH)

TOP VIEW  
(Back Edge of PC Board)

PINOUT

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+/O SELECT</td>
</tr>
<tr>
<td>2</td>
<td>A0</td>
</tr>
<tr>
<td>3</td>
<td>A1</td>
</tr>
<tr>
<td>4</td>
<td>A2</td>
</tr>
<tr>
<td>5</td>
<td>A3</td>
</tr>
<tr>
<td>6</td>
<td>A4</td>
</tr>
<tr>
<td>7</td>
<td>A5</td>
</tr>
<tr>
<td>8</td>
<td>A6</td>
</tr>
<tr>
<td>9</td>
<td>A7</td>
</tr>
<tr>
<td>10</td>
<td>A8</td>
</tr>
<tr>
<td>11</td>
<td>A9</td>
</tr>
<tr>
<td>12</td>
<td>A10</td>
</tr>
<tr>
<td>13</td>
<td>A11</td>
</tr>
<tr>
<td>14</td>
<td>A12</td>
</tr>
<tr>
<td>15</td>
<td>A13</td>
</tr>
<tr>
<td>16</td>
<td>A14</td>
</tr>
<tr>
<td>17</td>
<td>A15</td>
</tr>
<tr>
<td>18</td>
<td>R/W</td>
</tr>
<tr>
<td>19</td>
<td>N.C.</td>
</tr>
<tr>
<td>20</td>
<td>I/O STROBE</td>
</tr>
<tr>
<td>21</td>
<td>RDY</td>
</tr>
<tr>
<td>22</td>
<td>DMA</td>
</tr>
<tr>
<td>23</td>
<td>INT OUT</td>
</tr>
<tr>
<td>24</td>
<td>DMA OUT</td>
</tr>
<tr>
<td>25</td>
<td>+5V</td>
</tr>
<tr>
<td>26</td>
<td>GND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>-12V</td>
</tr>
<tr>
<td>34</td>
<td>-5V</td>
</tr>
<tr>
<td>35</td>
<td>N.C.</td>
</tr>
<tr>
<td>36</td>
<td>7M</td>
</tr>
<tr>
<td>37</td>
<td>Q3</td>
</tr>
<tr>
<td>38</td>
<td>Q1</td>
</tr>
<tr>
<td></td>
<td>USER 1</td>
</tr>
<tr>
<td>41</td>
<td>DEVICE SELECT</td>
</tr>
<tr>
<td>42</td>
<td>D7</td>
</tr>
<tr>
<td>43</td>
<td>D6</td>
</tr>
<tr>
<td>44</td>
<td>D5</td>
</tr>
<tr>
<td>45</td>
<td>D4</td>
</tr>
<tr>
<td>46</td>
<td>D3</td>
</tr>
<tr>
<td>47</td>
<td>D2</td>
</tr>
<tr>
<td>48</td>
<td>D1</td>
</tr>
<tr>
<td>49</td>
<td>D0</td>
</tr>
<tr>
<td>50</td>
<td>+12V</td>
</tr>
</tbody>
</table>

(Toward Front Edge of PC Board)

LOCATIONS J2 TO J12

Figure 5  POWER CONNECTOR

TOP VIEW  
(Toward Right Side of PC Board)

PINOUT  
(BLUE/WHITE WIRE) -12V
(ORANGE WIRE) +5V
(BLACK WIRE) GND

-5V (BLUE WIRE)
+5V (BLUE WIRE)
+12V (ORANGE/WHITE WIRE)
GND (BLACK WIRE)

LOCATION K1
**SPEAKER CONNECTOR**

This is a MOLEX KK 1ØØ series connector with two .25" square pins on .1Ø" centers. See location and pin out in Figures 1 and 6.

**SIGNAL DESCRIPTION FOR SPEAKER**

+5V: System +5 Volts

SPKR: Output line to speaker. Will deliver about .5 watt into 8 Ohms.

---

**VIDEO OUTPUT JACK**

This standard RCA phono jack located at the back edge of the APPLE II P.C. board will supply NTSC compatible, EIA standard, positive composite video to an external video monitor.

A video level control near the connector allows the output level to be adjusted from Ø to 1 Volt (peak) into an external 75 OHM load.

Additional tint (hue) range is provided by an adjustable trimmer capacitor.

See locations illustrated in Figure 1.
AUXILIARY VIDEO OUTPUT CONNECTOR

This is a MOLEX KK 100 series connector with four .25" square pins on .10" centers. It provides composite video and two power supply voltages. Video out on this connector is not adjustable by the on board 200 Ohm trim pot. See Figures 1 and 7.

SIGNAL DESCRIPTION

GND: System circuit ground. 0 Volt line from power supply.

VIDEO NTSC compatible positive composite VIDEO. DC coupled emitter follower output (not short circuit protected). SYNC TIP is 0 Volts, black level is about .75 Volts, and white level is about 2.0 Volts into 470 Ohms. Output level is non-adjustable.

+12V: +12 Volt line from power supply.

+5V: -5 Volt line from power supply.

Figure 7 AUXILIARY VIDEO OUTPUT CONNECTOR

PINOUT

+12V

-5V

VIDEO

GND

Right Edge of PC Board

Back Edge of PC Board

LOCATION J14B
INSTALLING YOUR OWN RAM

THE POSSIBILITIES

The APPLE II computer is designed to use dynamic RAM chips organized as 4096 x 1 bit, or 16384 x 1 bit called "4K" and "16K" RAMs respectively. These must be used in sets of 8 to match the system data bus (which is 8 bits wide) and are organized into rows of 8. Thus, each row may contain either 4096 (4K) or 16384 (16K) locations of Random Access Memory depending upon whether 4K or 16K chips are used. If all three rows on the APPLE II board are filled with 4K RAM chips, then 12288 (12K) memory locations will be available for storing programs or data, and if all three rows contain 16K RAM chips then 49152 (commonly called 48K) locations of RAM memory will exist on board!

RESTRICTIONS

It is quite possible to have the three rows of RAM sockets filled with any combination of 4K RAMs, 16K RAMs or empty as long as certain rules are followed:

1. All sockets in a row must have the same type (4K or 16K) RAMs.
2. There MUST be RAM assigned to the zero block of addresses.

ASSIGNING RAM

The APPLE II has 48K addresses available for assignment of RAM memory. Since RAM can be installed in increments as small as 4K, a means of selecting which address range each row of memory chips will respond to has been provided by the inclusion of three MEMORY SELECT sockets on board.

Figure 8
MEMORY SELECT SOCKETS

<table>
<thead>
<tr>
<th>PINOUT</th>
<th>TOP VIEW</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0000-0FFF) 4K &quot;0&quot; BLOCK</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1000-1FFF) 4K &quot;1&quot; BLOCK</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2000-2FFF) 4K &quot;2&quot; BLOCK</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3000-3FFF) 4K &quot;3&quot; BLOCK</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4000-4FFF) 4K &quot;4&quot; BLOCK</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(5000-5FFF) 4K &quot;5&quot; BLOCK</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(6000-6FFF) 4K &quot;6&quot; BLOCK</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LOCATIONS D1, E1, F1
INTRODUCTION

APPLE II is supplied completely tested with the specified amount of RAM memory and correct memory select jumpers. There are five different sets of standard memory jumper blocks:

1. 4K 4K 4K BASIC
2. 4K 4K 4K HIRES
3. 16K 4K 4K
4. 16K 16K 4K
5. 16K 16K 16K

A set of three each of one of the above is supplied with the board. Type 1 is supplied with 4K or 8K systems. Both type 1 and 2 are supplied with 12K systems. Type 1 is a contiguous memory range for maximum BASIC program size. Type 2 is non-contiguous and allows 8K dedicated to HIRES screen memory with approximately 2K of user BASIC space. Type 3 is supplied with 16K, 2C0K and 24K systems. Type 4 with 30K and 36K systems and type 5 with 48K systems.

Additional memory may easily be added just by plugging into sockets along with correct memory jumper blocks.

The 6502 microprocessor generates a 16 bit address, which allows 65536 (commonly called 65K) different memory locations to be specified. For convenience we represent each 16 bit (binary) address as a 4-digit hexadecimal number. Hexadecimal notation (hex) is explained in the Monitor section of this manual.

In the APPLE II, certain address ranges have been assigned to RAM memory, ROM memory, the I/O bus, and hardware functions. The memory and address maps give the details.
MEMORY SELECT SOCKETS

The location and pin out for memory select sockets are illustrated in Figures 1 and 8.

HOW TO USE

There are three MEMORY SELECT sockets, located at D1, E1 and F1 respectively. RAM memory is assigned to various address ranges by inserting jumper wires as described below. All three MEMORY SELECT sockets MUST be jumpered identically! The easiest way to do this is to use Apple supplied memory blocks.

Let us learn by example:

If you have plugged 16K RAMs into row "C" (the sockets located at C3-C10 on the board), and you want them to occupy the first 16K of addresses starting at 0000, jumper pin 14 to pin 10 on all three MEMORY SELECT sockets (thereby assigning row "C" to the 0000-3FFF range of memory).

If in addition you have inserted 4K RAMs into rows "D" and "E", and you want them each to occupy the first 4K addresses starting at 4000 and 5000 respectively, jumper pin 13 to pin 5 (thereby assigning row "D" to the 4000-4FFF range of memory), and jumper pin 12 to pin 6 (thereby assigning row "E" to the 5000-5FFF range of memory). Remember to jumper all three MEMORY SELECT sockets the same.

Now you have a large contiguous range of addresses filled with RAM memory. This is the 24K addresses from 0000-5FFF.

By following the above examples you should be able to assign each row of RAM to any address range allowed on the MEMORY SELECT sockets. Remember that to do this properly you must know three things:

1. Which rows have RAM installed?
2. Which address ranges do you want them to occupy?
3. Jumper all three MEMORY SELECT sockets the same!

If you are not sure think carefully, essentially all the necessary information is given above.
Memory Address Allocations in 4K Bytes

<table>
<thead>
<tr>
<th>HEX ADDRESS (ES)</th>
<th>USED BY</th>
<th>USED FOR</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGE ZERO 0000-001F</td>
<td>UTILITY</td>
<td>register area for &quot;sweet 16&quot; 16 bit firmware processor.</td>
<td></td>
</tr>
<tr>
<td>0020-004D</td>
<td>MONITOR</td>
<td>holds a 16 bit number that is randomized with each key entry.</td>
<td></td>
</tr>
<tr>
<td>004E-004F</td>
<td>MONITOR</td>
<td>integer multiply and divide work space.</td>
<td></td>
</tr>
<tr>
<td>0050-0055</td>
<td>UTILITY</td>
<td>floating point work space.</td>
<td></td>
</tr>
<tr>
<td>0055-00FF</td>
<td>BASIC</td>
<td>subroutine return stack.</td>
<td></td>
</tr>
<tr>
<td>00FF-00FF</td>
<td>UTILITY</td>
<td>character input buffer.</td>
<td></td>
</tr>
<tr>
<td>PAGE ONE 0100-01FF</td>
<td>6502</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAGE TWO 0200-02FF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAGE THREE 03FF</td>
<td>MONITOR</td>
<td>Y (control Y) will cause a JSR to this location.</td>
<td></td>
</tr>
<tr>
<td>03FB</td>
<td></td>
<td>MMI's are vectored to this location.</td>
<td></td>
</tr>
<tr>
<td>03FE-03FF</td>
<td></td>
<td>IRQ's are vectored to the address pointed to by these locations.</td>
<td></td>
</tr>
<tr>
<td>0400-07FF</td>
<td>DISPLAY</td>
<td>text or color graphics primary page.</td>
<td></td>
</tr>
<tr>
<td>0800-0BFF</td>
<td>DISPLAY</td>
<td>text or color graphics secondary page.</td>
<td></td>
</tr>
</tbody>
</table>

Memory Map Pages 0 to BFF

- Addresses dedicated to hardware functions
- ROM socket D0: spare
- ROM socket D8: spare
- ROM socket E0: BASIC
- ROM socket E8: BASIC
- ROM socket F0: BASIC
- ROM socket F8: monitor

BASIC initializes LOADM to location 0800.
<table>
<thead>
<tr>
<th>HEX ADDRESS</th>
<th>ASSIGNED FUNCTION</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C00X</td>
<td>Keyboard input.</td>
<td>Keyboard strobe appears in bit 7. ASCII data from keyboard appears in the 7 lower bits.</td>
</tr>
<tr>
<td>C01X</td>
<td>Clear keyboard strobe.</td>
<td></td>
</tr>
<tr>
<td>C02X</td>
<td>Toggle cassette output.</td>
<td></td>
</tr>
<tr>
<td>C03X</td>
<td>Toggle speaker output.</td>
<td></td>
</tr>
<tr>
<td>C04X</td>
<td>&quot;CO40 STB&quot;</td>
<td>Output strobe to Game I/O connector.</td>
</tr>
<tr>
<td>C050</td>
<td>Set graphics mode</td>
<td></td>
</tr>
<tr>
<td>C051</td>
<td>&quot; text &quot;</td>
<td></td>
</tr>
<tr>
<td>C052</td>
<td>Set bottom 4 lines graphics</td>
<td></td>
</tr>
<tr>
<td>C053</td>
<td>&quot; &quot; &quot; &quot; text</td>
<td></td>
</tr>
<tr>
<td>C054</td>
<td>Display primary page</td>
<td></td>
</tr>
<tr>
<td>C055</td>
<td>&quot; secondary page</td>
<td></td>
</tr>
<tr>
<td>C056</td>
<td>Set high res. graphics</td>
<td></td>
</tr>
<tr>
<td>C057</td>
<td>&quot; color &quot;</td>
<td></td>
</tr>
<tr>
<td>C058</td>
<td>Clear &quot;ANO&quot;</td>
<td>Annunciator 0 output to Game I/O connector.</td>
</tr>
<tr>
<td>C059</td>
<td>Set &quot;</td>
<td></td>
</tr>
<tr>
<td>C05A</td>
<td>Clear &quot;AN1&quot;</td>
<td>Annunciator 1 output to Game I/O connector.</td>
</tr>
<tr>
<td>C05B</td>
<td>Set &quot;</td>
<td></td>
</tr>
<tr>
<td>C05C</td>
<td>Clear &quot;AN2&quot;</td>
<td>Annunciator 2 output to Game I/O connector.</td>
</tr>
<tr>
<td>C05D</td>
<td>Set &quot;</td>
<td></td>
</tr>
<tr>
<td>C05E</td>
<td>Clear &quot;AN3&quot;</td>
<td>Annunciator 3 output to Game I/O connector.</td>
</tr>
<tr>
<td>C05F</td>
<td>Set &quot;</td>
<td></td>
</tr>
<tr>
<td>HEX ADDRESS</td>
<td>ASSIGNED FUNCTION</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------</td>
<td>----------</td>
</tr>
<tr>
<td>C060/8</td>
<td>Cassette input</td>
<td>State of &quot;Cassette Data In&quot; appears in bit 7. input on State of Switch 1 Game I/O connector appears in bit 7.</td>
</tr>
<tr>
<td>C061/9</td>
<td>&quot;SW1&quot;</td>
<td>State of Switch 2 input on Game I/O connector appears in bit 7.</td>
</tr>
<tr>
<td>C062/A</td>
<td>&quot;SW2&quot;</td>
<td>State of Switch 3 input on Game I/O connector appears in bit 7.</td>
</tr>
<tr>
<td>C063/B</td>
<td>&quot;SW3&quot;</td>
<td>State of timer output for Paddle 0 appears in bit 7.</td>
</tr>
<tr>
<td>C064/C</td>
<td>Paddle 0 timer output</td>
<td>State of timer output for Paddle 1 appears in bit 7.</td>
</tr>
<tr>
<td>C065/D</td>
<td>&quot; 1 &quot; &quot; &quot;</td>
<td>State of timer output for Paddle 2 appears in bit 7.</td>
</tr>
<tr>
<td>C066/E</td>
<td>&quot; 2 &quot; &quot; &quot;</td>
<td>State of timer output for Paddle 3 appears in bit 7.</td>
</tr>
<tr>
<td>C067/F</td>
<td>&quot; 3 &quot; &quot; &quot;</td>
<td>Triggers paddle timers during Φ₂.</td>
</tr>
<tr>
<td>C07X</td>
<td>&quot;PDL STB&quot;</td>
<td>Pin 41 on the selected Peripheral Connector goes low during Φ₂.</td>
</tr>
<tr>
<td>C08X</td>
<td>DEVICE SELECT 0</td>
<td>Expansion connectors.</td>
</tr>
<tr>
<td>C09X</td>
<td>&quot; 1 &quot;</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>C0AX</td>
<td>&quot; 2 &quot;</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>C0BX</td>
<td>&quot; 3 &quot;</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>C0CX</td>
<td>&quot; 4 &quot;</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>C0DX</td>
<td>&quot; 5 &quot;</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>C0EX</td>
<td>&quot; 6 &quot;</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>C0FX</td>
<td>&quot; 7 &quot;</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>C10X</td>
<td>&quot; 8 &quot;</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>C11X</td>
<td>&quot; 9 &quot;</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>C12X</td>
<td>&quot; A &quot;</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>HEX ADDRESS</td>
<td>ASSIGNED FUNCTION</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------</td>
<td>----------</td>
</tr>
<tr>
<td>C13X</td>
<td>DEVICE SELECT</td>
<td>B</td>
</tr>
<tr>
<td>C14X</td>
<td>&quot;</td>
<td>C</td>
</tr>
<tr>
<td>C15X</td>
<td>&quot;</td>
<td>D</td>
</tr>
<tr>
<td>C16X</td>
<td>&quot;</td>
<td>E</td>
</tr>
<tr>
<td>C17X</td>
<td>&quot;</td>
<td>F</td>
</tr>
<tr>
<td>C1XX</td>
<td>I/O SELECT</td>
<td>1</td>
</tr>
<tr>
<td>C2XX</td>
<td>&quot;</td>
<td>2</td>
</tr>
<tr>
<td>C3XX</td>
<td>&quot;</td>
<td>3</td>
</tr>
<tr>
<td>C4XX</td>
<td>&quot;</td>
<td>4</td>
</tr>
<tr>
<td>C5XX</td>
<td>&quot;</td>
<td>5</td>
</tr>
<tr>
<td>C6XX</td>
<td>&quot;</td>
<td>6</td>
</tr>
<tr>
<td>C7XX</td>
<td>&quot;</td>
<td>7</td>
</tr>
<tr>
<td>C8XX</td>
<td>&quot;</td>
<td>8, I/O STROBE</td>
</tr>
<tr>
<td>C9XX</td>
<td>&quot;</td>
<td>9,</td>
</tr>
<tr>
<td>CAXX</td>
<td>&quot;</td>
<td>A,</td>
</tr>
<tr>
<td>CBXX</td>
<td>&quot;</td>
<td>B,</td>
</tr>
<tr>
<td>CCXX</td>
<td>&quot;</td>
<td>C,</td>
</tr>
<tr>
<td>CDXX</td>
<td>&quot;</td>
<td>D,</td>
</tr>
<tr>
<td>CEXX</td>
<td>&quot;</td>
<td>E,</td>
</tr>
<tr>
<td>CFXX</td>
<td>&quot;</td>
<td>F,</td>
</tr>
<tr>
<td>D000-D7FF</td>
<td>ROM socket D0</td>
<td></td>
</tr>
<tr>
<td>D800-DFFF</td>
<td>&quot;</td>
<td>D8</td>
</tr>
<tr>
<td>E000-E7FF</td>
<td>&quot;</td>
<td>E0</td>
</tr>
<tr>
<td>E800-EFFF</td>
<td>&quot;</td>
<td>E8</td>
</tr>
<tr>
<td>F000-F7FF</td>
<td>&quot;</td>
<td>F0</td>
</tr>
<tr>
<td>F800-FFFF</td>
<td>&quot;</td>
<td>F8</td>
</tr>
</tbody>
</table>

NOTES:
1. Peripheral Connector 0 does not get this signal.
2. I/O SELECT 1 uses the same addresses as DEVICE SELECT 8-F.

Expansion connectors.

Spare.
Spare.
BASIC.
BASIC.
1K of BASIC, 1K of utility.
Monitor.
SYSTEM TIMING

SIGNAL DESCRIPTIONS

14M: Master oscillator output, 14.318 MHz +/- 35 ppm. All other timing signals are derived from this one.

7M: Intermediate timing signal, 7.159 MHz.

COLOR REF: Color reference frequency used by video circuitry, 3.530 MHz.

Ø0: Phase 0 clock to microprocessor, 1.023 MHz nominal.

Ø1: Microprocessor phase 1 clock, complement of Ø0, 1.023 MHz nominal.

Ø2: Same as Ø0. Included here because the 6502 hardware and programming manuals use the designation Ø2 instead of Ø0.

Q3: A general purpose timing signal which occurs at the same rate as the microprocessor clocks but is nonsymmetrical.

MICROPROCESSOR OPERATIONS

ADDRESS: The address from the microprocessor changes during Ø1, and is stable about 300nS after the start of Ø1.

DATA WRITE: During a write cycle, data from the microprocessor appears on the data bus during Ø2, and is stable about 300nS after the start of Ø2.

DATA READ: During a read cycle, the microprocessor will expect data to appear on the data bus no less than 100nS prior to the end of Ø2.

SYSTEM TIMING DIAGRAM
FIGURE S-1  APPLE II SYSTEM DIAGRAM
FIGURE S-2 MPU AND SYSTEM BUS
FIGURE S-3  REFERENCE OSCILLATOR AND SYSTEM TIMING
FIGURE S-4  SYNC COUNTER
FIGURE S-7 RAM ADDRESS MUX

*SEE FIG. S-6 FOR OTHER HALF OF C12
FIGURE S-8 4K TO 48K RAM MEMORY WITH DATA LATCH
FIGURE S-9 PERIPHERAL I/O CONNECTOR PINOUT AND CONTROL LOGIC
Figure S-11 Video Generator
This recreation of the 1978 Apple II RedBook is courtesy of gerrydoire@yahoo.ca

If you plan to keep it for any use please donate for better software and computer hardware. Donations can be made to paypal by using the following link: marketplace@seaside.ns.ca

PayPal

Software currently in need of revision for accuracy and speed: Adobe Illustrator, Adobe Photoshop, Adobe Acrobat Pro, etc.