This recreation of the 1978 Apple ][ Redbook is courtesy of Gerry Doire. 
gerrydoire@yahoo.ca for any comments or suggestions.

If you wish to donate monetary or computer hardware 
for the continuation of this project email: gerrydoire@yahoo.ca

PayPal

Current Donators to this project are: 
Dan Chisarick, David Schmidt
Apple II
Reference Manual
January 1978
APPLE II Reference Manual

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 Dis-assembler 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 112
3. Interfacing with the Home TV 114
4. Simple Serial Output 115
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
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:

A2S00MMX

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

A2S008X

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 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

5 GOTO 15
10 O<( PDL (8)-20)/6: IF O<0 THEN
0: IF O>=34 THEN O=34: COLOR= D: VLIM 0.0-5 AT 8: COLOR=A:
IF P>0 THEN 175: IF O THEN
VLIM 0.0-1 AT 8: P=O:RETURN
15 DIM A$(15),B$(10):A=1:B=13:
C=9:D=6:E=15: TEXT : CALL -
936: VTAB 4: TAB 10: PRINT
'*** BREAKOUT ***':PRINT
20 PRINT "OBJECT IS TO DESTROY
ALL BRICKS": PRINT : INPUT
'HI, WHAT'S YOUR NAME? ',A$;
25 PRINT "STANDARD COLORS ";A$;
PRINT "Y/N? ":B$: GR: CALL
-936: IF B$(1,1)#"N" THEN 40
FOR I=0 TO 39: COLOR=I/2*
(I(32): VLIN 0,39 AT I
30 NEXT I:
POKE 34,20: PRINT :
PRINT :
FOR I=0 TO 15: VTAB 21+I MOD 2:
TAB I+I+1: PRINT I;: NEXT I:
POKE 34,22: YTAB 24: PRINT 
"BACKGROUND";
35 GOSUB 95:TAKE: PRINT "EVEN BRICK"
;GOSUB 95:TAKE: PRINT "ODD BRIC
K": GOSUB 95:TAKE: PRINT "PADDLE"
;: GOSUB 95:TAKE: PRINT "BALL"
;: GOSUB 95
40 POKE 34,28: COLOR=A: FOR I=
0 TO 39: VLIM 0.39 AT I: NEXT
I: FOR I=28 TO 34 STEP 2: TAB
1: PRINT I/2-9: COLOR=B:
VLIM 0.39 AT I: COLOR=C: FOR
J=I MOD 4 TO 39 STEP 4
45 YLIM J,J+1 AT I: NEXT J, I:
TAB 5: PRINT "SCORE=":PRINT
5: PRINT "LAST BALL, ":;
A$: PRINT FOR I=1 TO 100:
GOSUB 16: NEXT I:
PRINT "EVEN BRICK";
50 GOSUB 95:A=E: PRINT "ODD BRICK";
GOSUB 95:B=E: PRINT "PADDLE"
;GOSUB 95:C=E: PRINT "BALL"
;GOSUB 95
55 IF L=1 THEN PRINT "LAST BALL, ";
;A$: PRINT FOR I=1 TO 100:
GOSUB 16: NEXT I:
PRINT "HELP!": PRINT
60 FOR I=0 TO 39: COLOR=I/2*-(
I(32): VLIN 0,39 AT I
65 NEXT I:
POKE 34,21: S=0: P=S:
X=19: Y=19: L=6
50 COLOR=3: PLOT K,Y/3:X=19;Y=
RND (128)*:Y=-1111: RND (5)
2: L=L-1: IF L<1 THEN 120:
TAB 6: IF L>1 THEN PRINT L;
"BALLS LEFT"
70 IF L=1 THEN PRINT "LAST BALL, ";
;A$: PRINT FOR I=1 TO 100:
GOSUB 16: NEXT I:
PRINT "HELP!": PRINT
75 FOR I=1 TO 6: M=PEEK (-16336):
NEXT I:
I=X: M=0
80 V=-V
85 PLOT X,Y/3: COLOR=E: PLOT I,
K:X=I:Y=J: GOTO 60
90 PRINT "INVALID, REENTER";
95 INPUT " COLOR (0 TO 15)",E:
IF E<0 OR E>15 THEN 90:
RETURN
100 IF M THEN V=ABS (V): YLIM
K/2+2,K/2+2 AT I: S=S+1/2-
105 O=PEEK (-16386) PEEK (-16386)
) PEEK (-16386) PRINT (-16386)
) PEEK (-16386) PRINT (-16386)
) PEEK (-16386) PRINT (-16386)
) PEEK (-16386) PRINT (-16386)
110 IF S<720 THEN 80
115 PRINT "CONGRATULATIONS, ";A$;
;" YOU WIN!": GOTO 165
120 PRINT "YOUR SCORE OF ";S;" IS 
;GOTO 125+(S/100)*5
125 PRINT "TERRIBLE!": GOTO 165
130 PRINT "TERRIBLE!": GOTO 165
135 PRINT "POOR.": GOTO 165
140 PRINT "FAIR.": GOTO 165
145 PRINT "GOOD.": GOTO 165
150 PRINT "EXCELLENT.": GOTO 165
155 PRINT "EXCELLENT.": GOTO 165
160 PRINT "HEATHLY PERFECT."
;GOTO 165
165 PRINT "ANOTHER GAME ";A$: Y/Y
;INPUT A$: IF A$(1,1)="Y"
THEN 25: TEXT : CALL -936:
YTAB 10: TAB 10: PRINT "GAME OV
ER": END
170 O<( PDL (8)-20)/6: IF O<0 THEN
0: IF O>=34 THEN O=34: COLOR= D: VLIM 0.0-5 AT 8: COLOR=A:
IF P>0 THEN 175: IF O THEN
VLIM 0.0-1 AT 8: P=O:RETURN
175 IF P>0 THEN RETURN : IF OH34
THEN VLIM 0.0-39 AT 8: P=O:
RETURN
180 FOR I=1 TO 88:OE=PEEK (-16386)
NEXT I: GOTO 50
COLOR DEMO PROGRAM LISTING

PROGRAM LISTING

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

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

30 TEXT ; CALL -996; VTAB 4; TAB 6; PRINT "4K COLOR DEMOS"; PRINT
: PRINT "1 LINES"; PRINT "2 CROS
5": PRINT "3 WEAVING"

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

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

60 PRINT "HIT ANY KEY FOR NEW DEMO"
: z=0; PRINT "INPUT " WHICH DEMO"
# " ; i = GR : IF i > 10 AND i < 11
: THEN GOTO 100; GOTO 30

70 INPUT "WHICH DEMO WOULD YOU LIKE"
: i = GR : IF i AND i > 20 THEN
: GOTO 100; GOTO 30

100 i=i+1 MOD 79;i=i<(1339)+(79
:-i); GOSUB 2000: GOSUB 10000
: GOTO 100

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

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

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

410 FOR i=1 TO 1 STEP 1:(C(I)
: =C(I): NEXT I:C(I)= RND (16)
: NEXT I:

420 COLOR=C(j);L=x+5+14+1;x=39-
: L: HLIN X,L AT K: VLIN X,L AT
: L: HLIN K,L AT L: VLIN K,L AT
: K: NEXT J:1; GOSUB 10000: GOTO
410

500 Z=2=z: GOTO 900

600 COLOR= RND (16): FOR i=0 TO
: 18 STEP 2;I=39-I: HLIN I,J AT
: I; GOSUB 648: VLIN I,J AT J;
GOSUB 648

610 HLIN I+2,J AT J: GOSUB 648:

630 VLIN I,J AT J: GOSUB 648: HLIN
I,J AT I: GOSUB 648: NEXT I:
GOSUB 10000: GOTO 600

640 K=x+7;L=x+5+4+26+70=3;2767
: L=L\( P L(X)\)\( P L(K)
: POKE 1,L MOD 256: POKE 24,
L\( P L(0)\)\( P L(K)
: CALL 2: RETURN

700 I=RND (39)+3: J=I*1+5+1:26+

701: RND (6): CALL 2

702 POKE 24,(K*255)+1: CALL 2

800 X=3;7=20000=P+1=280:4:4:Y=0

900 I=1: COLOR=6: HLIN 0,39 AT
: 4: COLOR=9: GOSUB 880: COLOR=

10: VLIN 5, 39 AT X

110 N=2*R-P/R/W: COLOR=0: GOSUB

880: VLIN 5, 39 AT X: X=X+1: IF

120 THEN 880: X=3; VLIN 5, 39

130 AT I: VLIN 5, 39 AT 2

140 P=R:R=4:Y=100: COLOR=12: GOSUB

880: COLOR=9: GOSUB 5, 39 AT
X: COLOR=15: PLOT X-2,M: FOR

150 I=0 TO J; NEXT I: GOSUB 10000

160 GOTO 810

170 M=L-Y-L1=M-1:2=M+1: VLIN L1,

180 L2 AT X-1: VLIN LI,L2 AT X:

190 RETURN

200 I=1+I MOD 15: FOR i=0 TO 33

210 : FOR X=0 TO 39: COLOR=1+(A B S

220 -(29-X)-Z)*ABS (26-Y-Z)-25

230 : PLOT X,Y: NEXT X,Y: GOSUB

10000: GOTO 900

1100 CALL -996

1200 J=J+1 MOD 32: COLOR=J/2: VLIN

8, 39 AT 3: VTAB 21+(J/2) MOD
2: TAB 3+J: IF J MOD 2 THEN

130 PRINT J:2; GOSUB 10000: GOTO

140

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

39 AT J: RETURN

15000 IF PEEK (-16384)/128 THEN RETURN

16000 POKE -16388,0: 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.
   B. A 2 REGENERATES THE ENERGY AT THE Expense OF TIME.
   C. A 3 GIVES THE CONTENTS OF THE IMMEDIATE ADJACENT QUADRANTS.
   D. A 4 FIRES PHASERS AT THE Expense OF AVAILABLE ENERGY.
   E. A 5 INITIATES A SET OF QUESTIONS FOR TORPEDO FIRING.
   F. 6, 8 AND 10 ALL GIVE INFORMATION ABOUT THE STATUS OF THE SHIP AND ITS ENVIRONMENT.
   G. 9 SETS THE SHIELD ENERGY/AVAILABLE ENERGY RATIO.
   H. 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. 7 ENTERS A COMPUTER WHICH WILL responding TO THE FOLLOWING INSTRUCTIONS:

   1. COMPUTE COURSE 2. LOCK PHASERS
   3. LOCK PHOTON TORPEDOES 4. LOCK COURSE
   5. COMPUTE TREJECTORY 6. STATUS
   7. RETURN TO COMMAND MODE

   In THE FIFTH 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 TREJECTORY:

   0

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

   180

--- THIS EXPLANATION Was Written BY ELMWOOD ---
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 subprogram. 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.

COMMAND NAME

AUTO num
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 "MAN" and press the return key.

AUTO num1,num2
Same as above except increments line numbers by number num2.

CLR
Clears current BASIC variables; undimensions arrays. Program is unchanged.

CON
Continues program execution after a stop from a control C*. Does not change variables.

DEL num1,
Deletes line number num1.

DEL num1,num2
Deletes program from line number num1 through line number num2.

DSP var
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.)

HIMEM expr
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*.

GOTO expr
Causes immediate jump to line number specified by expression expr.

GR
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.

LIST
Lists entire program on screen.

LIST num1
Lists program line number num1.

LIST num1,num2
Lists program line number num1 through line number num2.
**LOAD** *expr.*  
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:  
destroy 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  
exectes 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>1Ø X= 4*(5 + X)</td>
<td>Expressions within parenthesis ( ) are always evaluated first.</td>
</tr>
<tr>
<td>+</td>
<td>2Ø X= 1+4*5</td>
<td>Optional; +l times following expression.</td>
</tr>
<tr>
<td>-</td>
<td>3Ø ALPHA = -(BETA +2)</td>
<td>Negation of following expression.</td>
</tr>
<tr>
<td>NOT</td>
<td>4Ø IF A NOT B THEN 200</td>
<td>Logical Negation of following expression; Ø if expression is true (non-zero), l if expression is false (zero).</td>
</tr>
</tbody>
</table>

| **Arithmetic Operators** | | |
| † | 6Ø Y = X 3 | Exponentiate as in $X^3$. NOTE: † is shifted letter N. |
| * | 7Ø LET DOTS=A*B*N2 | Multiplication. NOTE: Implied multiplication such as $(2 + 3)(4)$ is not allowed thus N2 in example is a variable not N * 2. |
| / | 8Ø PRINT GAMMA/S | Divide |
| MOD | 9Ø X = 12 MOD 7 | Modulo: Remainder after division of first expression by second expression. |
| | 10Ø X = X MOD(Y+2) | |
| + | 11Ø P = L + G | Add |
| - | 12Ø XY4 = H-D | Subtract |
| = | 13Ø HEIGHT=15 | Assignment operator; assigns a value to a variable. LET is optional |
| | 14Ø LET SIZE=7*5 | |
| | 15Ø A(8) = 2 | |
| | 155 ALPHA$ = "PLEASE" | |
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</td>
<td>Expression &quot;equals&quot; expression.</td>
</tr>
<tr>
<td></td>
<td>THEN 500</td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>170 IF A$(1,1)=&quot;Y&quot; THEN 500</td>
<td>String variable &quot;equal&quot; string variable.</td>
</tr>
<tr>
<td># or &lt; &gt;</td>
<td>180 IF ALPHA #*Y THEN 500</td>
<td>Expression &quot;does not equal&quot; expression.</td>
</tr>
<tr>
<td>#</td>
<td>190 IF A$ &quot;NO&quot;</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></td>
<td>THEN 500</td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>200 IF A&gt;B</td>
<td>Expression &quot;is greater than&quot; expression.</td>
</tr>
<tr>
<td></td>
<td>THEN GO TO 50</td>
<td></td>
</tr>
<tr>
<td>&lt;</td>
<td>210 IF A+1&lt;B-5</td>
<td>Expression &quot;is less than&quot; expression.</td>
</tr>
<tr>
<td></td>
<td>THEN 100</td>
<td></td>
</tr>
<tr>
<td>&gt;=</td>
<td>220 IF A&gt;=B</td>
<td>Expression &quot;is greater than or equal to&quot; expression.</td>
</tr>
<tr>
<td></td>
<td>THEN 100</td>
<td></td>
</tr>
<tr>
<td>&lt;=</td>
<td>230 IF A+1&lt;=B-6</td>
<td>Expression &quot;is less than or equal to&quot; expression.</td>
</tr>
<tr>
<td></td>
<td>THEN 200</td>
<td></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

ABS (expr) 300 PRINT ABS(X) Gives absolute value of the expression expr.
ASC (str$) 310 PRINT ASC("BACK") Gives decimal ASCII value of designated string variable str. If more than one character is in designated string or sub-string, it gives decimal ASCII value of first character.
LEN (str$) 340 PRINT LEN(B$) Gives current length of designated string variable str$; i.e., number of characters.
PDL (expr) 350 PRINT PDL(X) Gives number between Ø and 255 corresponding to paddle position on game paddle number designated by expression expr and must be legal paddle (Ø,1,2, or 3) or else 255 is returned.
PEEK (expr) 360 PRINT PEEK(X) Gives the decimal value of number stored of decimal memory location specified by expression expr. For MEMORY locations above 32676, use negative number; i.e., HEX location FFFØ is -16.
RND (expr) 370 PRINT RND(X) Gives random number between V and (expression expr -1) if expression expr is positive; if minus, it gives random number between Ø and (expression expr +1).
SCRN(expr1, expr2) 380 PRINT SCRN(X1,Y1) Gives color (number between Ø and 15) of screen at horizontal location designated by expression expr1 and vertical location designated by expression expr2. Range of expression expr1 is Ø to 39. Range of expression expr2 is Ø to 39 if in standard color graphics display mode as set by GR command or Ø to 47 if in all color mode set by POKE -163Ø4 ,Ø: POKE -163Ø2,Ø'.
SGN (expr) 390 PRINT SGN(X) Gives sign (not sine) of expression expr i.e., -1 if expression expr is negative, zero and +1 if expr is positive.
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 alphanumeric characters up to 100. Variable names may not contain 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 hexidecimal 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)  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  Sets debug mode that DSP variable var each time it changes and the line number where the change occured.

Illegal:  100 DSP AX,B  102 DSP AB$  104 DSP A(5)
<table>
<thead>
<tr>
<th>NAME</th>
<th>EXAMPLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>END</td>
<td>110 END</td>
<td>Stops program execution. Sends carriage return and &quot;&gt;&quot; BASIC prompt) to screen.</td>
</tr>
<tr>
<td>FOR</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 expr3 is omitted, a STEP of +1 is assumed. Negative numbers are allowed.</td>
</tr>
<tr>
<td></td>
<td>120 FOR X=Y1 TO Y3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130 FOR I=39 TO 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>150 GOSUB 100 *J2</td>
<td></td>
</tr>
<tr>
<td></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>GOTO</td>
<td>160 GOTO 200</td>
<td>Causes immediate jump to legal line number specified by expression expr.</td>
</tr>
<tr>
<td></td>
<td>170 GOTO ALPHA+100</td>
<td></td>
</tr>
<tr>
<td>GR</td>
<td>180 GR</td>
<td>Sets mixed standard resolution color graphics mode. Initializes COLOR = 0 (Black) for top 40x40 of screen and sets scrolling window to lines 21 through 24 by 40 characters for four lines of text at bottom of screen. Example 190 sets all color mode (40x48 field) with no text at bottom of screen.</td>
</tr>
<tr>
<td></td>
<td>190 GR: POKE -16302,0</td>
<td></td>
</tr>
<tr>
<td>HLIN</td>
<td>200 HLIN 0,39 AT 20</td>
<td>In standard resolution color graphics mode, this command draws a horizontal line of a predefined color (set by COLOR=) starting at horizontal position defined by expression expr1 and ending at position expr2 at vertical position defined by expression expr3. expr1, expr2, and expr3 must be in the range of 0 to 39 and expr1 &lt; expr2. expr3 must be in the range of 0 to 39 (or 0 to 47 if not in mixed mode).</td>
</tr>
<tr>
<td></td>
<td>210 HLIN 2,2+6 AT 1</td>
<td></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 20,39 AT 39 is a horizontal line at the bottom of the screen extending from center to right corner.
IF expression THEN statement

220 IF A > B THEN 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$

Enotes data into memory from I/O device. If number input is expected, APPLE will output "?"; if string input is expected no "?" will be outputed. 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 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, num2

350 IF X > 6 THEN LIST 50

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
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.

Stores decimal number defined by expression expr2 in range of 0 to 255 at decimal memory location specified by expression expr1. Locations above 32767 are specified by negative numbers.

"POPS" nested GOSUB return stack address by one.

Outputs data specified by variable var or string variable str$ starting at current cursor location. If there is not trailing "," or ";" (Ex 45Ø) a carriage return will be generated. Commas (Ex. 46Ø) outputs data in 5 left justified columns. Semi-colon (Ex. 47Ø) inhibits print of any spaces. Text imbedded in " " will be printed and may appear multiple times.

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

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

Causes branch to statement following last GOSUB; i.e., RETURN ends a subroutine. Do not confuse "RETURN" statement with Return key on keyboard.
\[ \text{TAB} \text{ expr} \]

530 TAB 24
540 TAB 1+24
550 IF A\#B THEN
   TAB 20

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

\[ \text{TEXT} \]

550 TEXT
560 TEXT: CALL-936

Sets all text mode. Resets scrolling window to 24 lines by 40 characters. Example 560 also clears screen and homes cursor to upper left corner.

\[ \text{TRACE} \]

570 TRACE
580 IFN >32000
   THEN TRACE

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

\[ \text{VLIN} \text{ expr1, expr2} \at \text{expr3} \]

590 VLIN 0, 39AT15
600 VLIN Z,Z+6ATY

Similar to HLIN except draws vertical line starting at \text{expr1} and ending at \text{expr2} at horizontal position \text{expr3}.

\[ \text{VTAB} \text{ expr} \]

610 VTAB 18
620 VTAB Z+2

Similar to TAB. Moves cursor to absolute vertical position specified by expression \text{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^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_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^C moves 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;÷&quot; on right side of keyboard that provides this functions 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 + 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>A</td>
<td>Move cursor to right</td>
</tr>
<tr>
<td>B</td>
<td>Move cursor to left</td>
</tr>
<tr>
<td>C</td>
<td>Move cursor down</td>
</tr>
<tr>
<td>D</td>
<td>Move cursor up</td>
</tr>
<tr>
<td>E</td>
<td>Clear text from cursor to end of line</td>
</tr>
<tr>
<td>F</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 Code</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

<table>
<thead>
<tr>
<th>Hex</th>
<th>BASIC Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Display Mode Controls</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CO50</td>
<td>10 POKE -16304.0</td>
<td>Set color graphics mode</td>
</tr>
<tr>
<td>CO51</td>
<td>20 POKE -16303.0</td>
<td>Set text mode</td>
</tr>
<tr>
<td>CO52</td>
<td>30 POKE -16302.0</td>
<td>Clear mixed graphics</td>
</tr>
<tr>
<td>CO53</td>
<td>40 POKE -16301.0</td>
<td>Set mixed graphics (4 lines text)</td>
</tr>
<tr>
<td>CO54</td>
<td>50 POKE -16300.0</td>
<td>Clear display Page. 2 (BASIC commands use Page 1 only)</td>
</tr>
<tr>
<td>CO55</td>
<td>60 POKE -16299.0</td>
<td>Set display to Page 2 (alternate)</td>
</tr>
<tr>
<td>CO56</td>
<td>70 POKE -16298.0</td>
<td>Clear HIRES graphics mode</td>
</tr>
<tr>
<td>CO57</td>
<td>80 POKE -16297.0</td>
<td>Set HIRES graphics mode</td>
</tr>
<tr>
<td><strong>TEXT Mode Controls</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0020</td>
<td>90 POKE 32,L1</td>
<td>Set left side of scrolling window to location specified by L1 in range of 0 to 39.</td>
</tr>
<tr>
<td>0021</td>
<td>100 POKE 33,W1</td>
<td>Set window width to amount specified by W1. L1+W1&lt;40. W1&gt;0</td>
</tr>
<tr>
<td>0022</td>
<td>110 POKE 34,T1</td>
<td>Set window top to line specified by T1 in range of 0 to 23</td>
</tr>
<tr>
<td>0023</td>
<td>120 POKE 35,B1</td>
<td>Set window bottom to line specified by B1 in the range of 0 to 23. B1&gt;T1</td>
</tr>
<tr>
<td>0024</td>
<td>130 CH=PEEK(36)</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>140 POKE 36,CH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>150 TAB(CH+1)</td>
<td></td>
</tr>
<tr>
<td>0025</td>
<td>160 CV=PEEK (37)</td>
<td>Similar to above. Read/set cursor vertical position in the range 0 to 23.</td>
</tr>
<tr>
<td></td>
<td>170 POKE 37,CV</td>
<td></td>
</tr>
<tr>
<td></td>
<td>180 VTAB(CV+1)</td>
<td></td>
</tr>
<tr>
<td>0032</td>
<td>190 POKE 50,127</td>
<td>Set inverse flag if 127 (Ex. 190)</td>
</tr>
<tr>
<td></td>
<td>200 POKE 50,255</td>
<td>Set normal flag if 255(Ex. 200)</td>
</tr>
<tr>
<td>FC58</td>
<td>210 CALL -936</td>
<td>(@E) Home cursor, clear screen</td>
</tr>
<tr>
<td>FC42</td>
<td>220 CALL -958</td>
<td>(F&lt;sub&gt;E&lt;/sub&gt;) Clear from cursor to end of page</td>
</tr>
<tr>
<td>Hex</td>
<td>BASIC Example</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>--------------------</td>
<td>-----------------------------------------------------</td>
</tr>
<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)</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 BASIC ERROR MESSAGES

*** SYNTAX ERR  Results from a syntactic or typing error.

*** > 32767 ERR  A value entered or calculated was less than -32767 or greater than 32767.

*** > 255 ERR  A value restricted to the range 0 to 255 was outside that range.

*** BAD BRANCH ERR  Results from an attempt to branch to a non-existent line number.

*** BAD RETURN ERR  Results from an attempt to execute more RETURNs than previously executed GOSUBs.

*** BAD NEXT ERR  Results from an attempt to execute a NEXT statement for which there was not a corresponding FOR statement.

*** 16 GOSUBS ERR  Results from more than 16 nested GOSUBs.

*** 16 FORS ERR  Results from more than 16 nested FOR loops.

*** NO END ERR  The last statement executed was not an END.

*** MEM FULL ERR  The memory needed for the program has exceeded the memory size allotted.

*** TOO LONG ERR  Results from more than 12 nested parentheses or more than 128 characters in input line.

*** DIM ERR  Results from an attempt to DIMension a string array which has been previously dimensioned.

*** RANGE ERR  An array was larger than the DIMensioned value or smaller than 1 or HLIN,VLIN, PLOT, TAB, or VTAB arguments are out of range.

*** STR OVFL ERR  The number of characters assigned to a string exceeded the DIMensioned value for that string.

*** STRING ERR  Results from an attempt to execute an illegal string operation.

RETYPE LINE  Results from illegal data being typed in response to an INPUT statement. This message also requests that the illegal item be retyped.
Simplified Memory Map

FFFF  64K  Monitor and BASIC Routines in ROM
E000  56K  Future enhancement or user supplied PROMs
D000  52K  Peripheral I/O
C000  48K  

XX  User specified RAM memory size
(HIMEM:)

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

LOMEM:
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:

\[
\text{YN} \quad \text{DSP} \quad \text{NVA} \quad \text{DATA(0)} \quad \text{DATA(1)} \ldots \text{DATA(N)}
\]

1 \hspace{1cm} h_1 \hspace{1cm} h_2 \hspace{1cm} h_{n+1}

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIABLE NAME</td>
<td>up to 100 characters represented in memory as ASCII equivalents with the high order bit set.</td>
</tr>
<tr>
<td>DSP (DISPLAY BYTE)</td>
<td>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.</td>
</tr>
<tr>
<td>NVA (NEXT VARIABLE ADDRESS)</td>
<td>two bytes (first low order, the second high order) indicating the memory location of the next variable.</td>
</tr>
<tr>
<td>DATA</td>
<td>hexadecimal equivalent of numeric information, represented in pairs of bytes, low order byte first.</td>
</tr>
</tbody>
</table>
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:

```
VN    DSP    NVA   DATA(0)   DATA(1)....   DATA(n)   ST
```

- **VN** - 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 la 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 lb) 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 table 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 Cl 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
- LM = 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.
<table>
<thead>
<tr>
<th>Var₁</th>
<th>Var₂</th>
<th>...</th>
<th>Varₙ</th>
<th>Unused Memory</th>
<th>P₁</th>
<th>P₂</th>
<th>P₃ ... Pₙ₋₂</th>
<th>Pₙ₋₁</th>
<th>Pₙ</th>
</tr>
</thead>
</table>

LOMEN: $800

CM End of Variable Table

PP beginning of Program

HIMEM Max System Size

b
Variable Data

a
BASIC Program

Figure 1

a) PRINT PEEK(204) + PEEK(205)\times256 \rightarrow PP

b) PRINT PEEK(202) + PEEK(203)\times256 \rightarrow CM

Figure 2

$800.80C$ rewritten with labelling

Figure 3
FIGURE 4b

READ/SAVE PROGRAM

0   A=Ø

This must be the first statement in the program. It is initially Ø, but if data is to be saved, it will equal the length of the data base.

10  GOTO 100

This statement moves command to the main program.

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

Lines 20-26 are the write data to tape subroutine.

22  A=CM-LM: POKE 60,4:
    POKE 61,8: POKE 62,5:
    POKE 63,8: CALL -307

Writing data table to tape

24  POKE 60,LM MOD 256:
    POKE 61, LM/256:
    POKE 62, CM MOD 256:
    POKE 63, CM/256:
    CALL -307

26  PRINT "DATA TABLE SAVED":
    RETURN

Returning control to main program.

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

Lines 30-38 are the READ data from tape subroutine.

32  POKE 60,4: POKE 61,8:
    POKE 62,5: POKE 63,8:
    CALL -259

34  IF A<0 THEN 38: P=LM+A:
    IF P>HM THEN 38: CM=P:
    POKE 60, LM MOD 256:
    POKE 61, LM/256: POKE 62,
    CM MOD 256: POKE 63, CM/256:
    CALL -259

Checking the record length (A) for memory requirements if everything is satisfactory the data is READ in.

36  PRINT "DATA READ IN":
    RETURN

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

Returning control to main program.

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

>LIST
0 A=0
10 GOTO 100
20 REM WRITE DATA TO TAPE ROUTINE
22 A=CM-LM: POKE 60,4: POKE 61,8: POKE 62,5: POKE 63,8: CALL -307
26 RETURN
30 REM READ DATA SUBROUTINE
32 POKE 60,4: POKE 61,8: POKE 62,5: POKE 63,8: CALL -259
36 RETURN
38 PRINT "*** TOO MUCH DATA BASE ***
**:END
110 PRINT '20 NUMBERS GENERATED'
120 PRINT 'NOW WE ARE GOING TO SAVE THE DATA': PRINT 'WHEN YOU ARE READY START THE RECORDER IN RECORD MORE': INPUT 'AND HIT RETURN': A$
130 CALL -936: PRINT 'NOW WRITING DATA TO TAPE': GSUB 20
135 PRINT 'NOW THE DATA IS SAVE'
140 PRINT 'NOW WE ARE GOING TO CLEAR THE X(20) TABLE AND READ THE DATA FROM TAPE'
150 FOR I=1 TO 20: X(I)=I: NEXT I
160 PRINT 'NOW START TAPE RECORDER': INPUT 'AND THEN HIT RETURN': A$
165 PRINT 'A',A
170 GSUB 30
180 PRINT 'ALL THE DATA READ IN'
190 FOR I=1 TO 20: PRINT 'X(',I,')=',X(I): NEXT I
195 PRINT 'THIS IS THE END'
200 END

100 DIM A$(1), X(20)
105 FOR I=1 TO 20: X(I)=I: NEXT I
108 LM=2048: CM=2106: A=58: HM=16383
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 $02 (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 (Ø). 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 $ØØ$ occurs. This process is repeated until
the tone has lasted a relative period of time equal to the duration (value in $Ø1$) 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 $Ø0$ and $Ø1$ 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.

FIGURE 2. BASIC "POKES"

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 C000 to FFF (hex). Equivalent addresses for the ROM subroutines will be in italic type face.
High-Resolution Operating Subroutines

**INIT** Initiates High-Resolution Graphics mode.
From BASIC: CALL 3072 (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 3886 (or CALL -12274)
From machine language: JSR SCOE (or JSR $L000E)

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 -21589)
From machine language: JSR $C7C (or JSR $L107C)

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 3786 (or CALL -11574)

From machine language: JSR $C95 (or JSR $D095)

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 3805 (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 806 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 of 16 will draw the shape rotated $90^\circ$ clockwise, etc. The rotation factor is passed in location 807 from BASIC or 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 $80$ byte, it is finished.

The three sections are arranged in a byte like this:

```
<table>
<thead>
<tr>
<th>7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>D D P O O O P O O</td>
</tr>
</tbody>
</table>
```

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 $\emptyset \emptyset$ (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 $\emptyset \emptyset$ at the end of a byte, then skip that section and go on to the next. When you have finished
SHAPE (continued)

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:**

![Diagram of shape vectors](image)

**Figure 1.**

**Codes:**

<table>
<thead>
<tr>
<th>Vector</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>000</td>
</tr>
<tr>
<td>→</td>
<td>010</td>
</tr>
<tr>
<td>↓</td>
<td>011</td>
</tr>
<tr>
<td>←</td>
<td>001</td>
</tr>
<tr>
<td>Empty</td>
<td>000</td>
</tr>
</tbody>
</table>

**Hex-decimal Codes**

<table>
<thead>
<tr>
<th>Vector</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>Empty</td>
<td>000</td>
</tr>
</tbody>
</table>

**Figure 2.**
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 GR.
105 FOR W=3 TO 50
110 FOR I=1 TO 19
115 FOR J=0 TO 19
120 K=I+J
130 COLOR=I+3/(I+3)+I*W/12
135 PLOT I,K: PLOT K,I: PLOT 40-I,40-K
136 PLOT 40-K,40-I: PLOT K,40-I:
140 NEXT J,I
145 NEXT W: GOTO 105
PROGRAM LISTING: PONG

5 10 10 REM PONG BY WENELL BITTER
15 REM 2/17/77
20 REM PARABLE Switches CONTROL
25 REM PARABLE SIZE AFTER A MISS
30 REM OR DURING A HIT
35 CR
40 CR
45 CR
50 CR
55 CR
60 CR
65 CR
70 CR
75 CR
80 CR
85 CR
90 CR
95 CR
100 CR
105 CR
110 CR
115 CR
120 CR
125 CR
130 CR
135 CR
140 CR
145 CR
150 CR
155 CR
160 CR
165 CR
170 CR
175 CR
180 CR
185 CR
190 CR
195 CR
200 CR
205 CR
210 CR
215 CR
220 CR
225 CR
230 CR
235 CR
240 CR
245 CR
250 CR
255 CR
260 CR
265 CR

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.
PROGRAM LISTING: COLOR SKETCH

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.
PROGRAM LISTING: MASTERMIND

A REI GAME OF MASTERMIND 8-25-77

REM (APPLE COMPUTER)
10 DIM MJ(4),J(4),K(4),O(4),X(4),A(4)
20 X(0)=0:K(0)=0:O(0)=0:J(0)=0:A(0)=0
40 A(1)=13:K(1)=13:O(1)=13:J(1)=13
60 X(1)=1:O(2)=8:J(2)=8:A(2)=8
80 X(2)=1:O(3)=8:J(3)=8:A(3)=8
100 REM CALL -004 HUE 3641:TRY=TRY+1:
120 FOR I=0 TO 3:PRINT 3641:TRY=TRY+1:
140 NEXT I:PRINT TRY:
160 COLOR=8:HL IN 0,20 AT T:FLASH=11 FOR
180 H=1 TO 8:GOSUB 1000:
200 NEXT H:
220 FOR I=1 TO 16:KEY=PEEK
240 (-16004):IF KEY=132 THEN 160
260 :POKE -16368,A:FLUSH=11 FOR
280 I=1 TO 3:IF KEY<>A THEN
300 :NEXT I:IF I=9 THEN
320 :FLUSH=11:KEY=165
340 :GOSUB 1000:
360 IF KEY=141 THEN 140
380 :POKE 14688,N+6:KEY=149 AND HK(6) THEN
400 :NEXT I:IF I=9 THEN
420 :GOTO 160
440 :GOTO 140
460 :FOR I=1 TO 5:
480 :NEXT I:COLOR=8:
500 :GOSUB 2000:NEXT I:
520 :IF N=5 THEN 500:COLOR=5:
540 :FOR I=1 TO 5:FOR H=1 TO
560 :GOSUB 2000:NEXT I:GOTO
580 REM PRINT = PRINT
590 REM CALL =004 TAB 7:PRINT
600 REM PRINT = "HIT ANY KEY TO BEGIN PLAY"

60 CALL -004: TAB 7:PRINT
62 "HIT ANY KEY TO BEGIN PLAY"
64 REM CALL -004: IF PEEK (-16004)
66 (132 THEN 160:POKE -16368,
68 A:PRINT = FOR I=1 TO
70 B(1)= AND (B(I+2 COLOR=5:
72 :B(1)=B(1)+1:16 AT 3a:PRINT
74 "N(1)): NEXT I
76 110 TRY=0: PRINT = "LETTER KEYS FOR COLOR CHANGE": PRINT
78 "ARROW KEYS FOR ARRANGE AND BACK":PRINT "HIT RETURN TO ADD EFT GUESS #":

600 REM CALL -004 SETS INVERSE V1D
602 REM CALL -004 SETS NORMAL V1D
604 REM PEEK (-16004) IS KEY (ASCII)
606 REM IF IT THEN STROBE SET
608 REM POKE -16368 CLORS AND STROBE
610 REM CALL-996 CLERAS SCREEN AND
612 REM CURSOR TO UPPER LEFT.
614 REM IN 31a, KEY=-68 OR +1
616 REM CURSOR=138 OR 140 ASCII
618 REM STATS 19-39 INTRD
620 REM STATS 100-110 NEW SETUP
622 REM STAT 268 NEW COLOR
624 REM STATS 300-310 USER INPUT
626 REM STAT 303 GUESS CRYE.
628 REM STAT 350-510 CYF
630 REM SUBR (800 COLOR LINE
632 REM SUBR 2000 MATCH TEST

650 REM PRINT = PRINT
660 REM CALL =004 TAB 5:PRINT
670 "YOU GOT IT IN "
680 TRY: PRINT "*: IF TRY>7 THEN PRINT "EXCELLENT"; IF TRY>
700 6 AND TRY>10 THEN PRINT "GOOD" ;
710 IF TRY>9 AND TRY<IS THEN PRINT
720 "AVERAGE"; IF TRY<7 THEN PRINT "POOR"; PRINT "); CALL
740 -004: TAB 5: PRINT "HIT ANY KEY
760 REM TO PLAY AGAIN": GOTO 100
780 REM IF H=6 THEN RETURN: COLOR=
800 :X:FLUSH=11:HL IN 4,2,A:
820 :4 AT Y: RETURN
840 REM IF X(I+2) THEN RETURN:
860 REM PRINT = PRINT "APRIL 1-3:6; PRINT ";
880 REM PRINT = PRINT "APRIL 1-3:6; RETURN
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 2,175: POKE 5,40: POKE 4,193: POKE 5,150: POKE 6,9
: POKE 7,32: POKE 8,150: POKE 9,250: POKE 10,150: POKE 11
: POKE 12,250: POKE 13,4: POKE 14,199: POKE 15,24: POKE
16,248: POKE 17,5: POKE 18,18: POKE 19,11: POKE 20,7: POKE
21,2: POKE 22,6: POKE 23,96:
15 GOTO 25
20 T=0: GOSUB 30: RETURN
25 PRINT "***************
**************": RETURN
30 XX=0: TOM=0:88: GOSUB 45: RETURN
35 XX=TOM=0:88: GOSUB 45: RETURN
40 XX=TOM=0:88: GOSUB 45:XX=9
:TOM=250: GOSUB 45: RETURN
45 POKE 1,356: POKE 256: POKE 24,
:TOM=256=11: POKE 3,XX: CALL 2:
RETURN
50 A=19:<FORE(1)=100>:PR=100:<
:FORE(1)=100>:PR=100:91=95
:FORE(1)=100>:PR=100:91=95
:FORE(1)=100>:PR=100:
55 A=0:PR=0:XX=1:Y=0:XX=1:<
:POKE 177:7=100:100:91=95
:POKE 177:7=100:100:91=95
:POKE 177:7=100:100:91=95:
RETURN
60 XX=0:XM=88: GOSUB 70:XX=9:
:TH=56: GOSUB 78: RETURN
65 XX=7: XM=18: GOSUB 78: RETURN

DRAGON MAZE PROGRAM

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.
1 TEXT : CALL -936
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 I
   AND THEN."
6 PRINT "TO MOVE, YOU HIT 'A' FOR RIGHT."
7 PRINT "'L' FOR LEFT, 'U' FOR UP, AND 'D' FOR DOWN. DO NOT HIT RETURN."
8 PRINT "THE OBJECT IS FOR YOU (THE GREEN DOT)."
9 PRINT "TO GET TO THE DOOR ON THE RIGHT SIDE."
10 PRINT "BEFORE THE DRAGON (THE RED DOT) EATS YOU."
11 PRINT "BEWARE!!!!!!!!! SOMETIMES THE DRAGON GETS REAL MAD, AND CLIMBS
   OVER A WALL."
12 PRINT "BUT MOST OF THE TIME, HE CAN'T DO OVER."n
13 PRINT "AND HAS TO GO AROUND."
14 PRINT "YOU CAN OFTEN 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 95 DIM M(3,3)
50 PRINT TYPE 'GO TO BEGIN' :
   INPUT @
160 OR = COLOR=15
165 CALL -935: PRINT "DRAGON MAZE:"
   PRINT (25); PRINT "GARY J. SHERMAN"
110 FOR I=0 TO 39 STEP 3: VLIN 8,39 AT I: NEXT
I
120 COLOR=0
125 S=1000
130 DIM H(39),T(39)
135 FOR I=1 TO 39: H(I)=RND: NEXT
I
140 FOR I=1 TO 39: T(I)=RND: NEXT
I
150 X=RND (13)+1; Y=RND (13)+1:
   @=189
160 IF C=1 THEN 1200
165 R=X+RND(3)+3; X=X+RND(Y-1)
   H(X)=H(X)+RND(1); T(X)=T(X)+RND(1)
170 IF Y=13 THEN 1200:X=RND(13)+1:
   Y=13
180 IF Y=13 THEN 1670:0=MXK(13):
   0
170 IF X=1 THEN 1890:0=MXK(13):
   0
1680 IF Y=1 THEN 1690:0=MXK(13):
   0
1580 @=R(0)+L(4)
1590 IF C=0 AND R(0) (10X2) OR
   0=0 THEN 1170
1110 OR= RND (4)
1220 GOTO 1130:1040
1330 IF NOT R THEN 1110:MXK=M(1)
   1+1
1435 VLIN 3+Y-2,3+Y-1 AT 3+X-1:
1535 GOTO 1035
1440 IF D=0 THEN 1110:MXK=M(1)
   +10=11:
1455 VLIN 3+X-2,3+X-1 AT 3+Y-1:
1460 GOTO 1035
1460 IF D=0 THEN 1110:MXK=M(1)
   1+1=1
1475 VLIN 3+Y-2,3+Y-1 AT 3+X:
1480 GOTO 1035
1480 IF D=0 THEN 1110:MXK=M(1)
   -1-1=1
1495 VLIN 3+X-2,3+X-1 AT 3+Y:
1505 GOTO 1035
1505 IF D=0 THEN 1110:MXK=M(1)
   1+1=1
1515 VLIN 3+Y-2,3+Y-1 AT 3+X:
1525 GOTO 1035
1525 IF D=0 THEN 1110:MXK=M(1)
   1+1=1
1535 VLIN 3+X-2,3+X-1 AT 3+Y:
1545 GOTO 1035
1545 IF D=0 THEN 1110:MXK=M(1)
   1+1=1
1555 VLIN 3+Y-2,3+Y-1 AT 3+X:
1565 GOTO 1035
1565 IF D=0 THEN 1110:MXK=M(1)
   1+1=1
1575 VLIN 3+X-2,3+X-1 AT 3+Y:
1585 GOTO 1035
1585 IF D=0 THEN 1110:MXK=M(1)
   1+1=1
1595 VLIN 3+Y-2,3+Y-1 AT 3+X:
1605 GOTO 1035
1605 IF D=0 THEN 1110:MXK=M(1)
   1+1=1
1615 VLIN 3+X-2,3+X-1 AT 3+Y:
1625 GOTO 1035
1625 IF D=0 THEN 1110:MXK=M(1)
   1+1=1
1635 VLIN 3+Y-2,3+Y-1 AT 3+X:
1645 GOTO 1035
1645 IF D=0 THEN 1110:MXK=M(1)
   1+1=1
1655 VLIN 3+X-2,3+X-1 AT 3+Y:
1665 GOTO 1035
1665 IF D=0 THEN 1110:MXK=M(1)
   1+1=1
1675 VLIN 3+Y-2,3+Y-1 AT 3+X:
1685 GOTO 1035
1685 IF D=0 THEN 1110:MXK=M(1)
   1+1=1
1695 VLIN 3+X-2,3+X-1 AT 3+Y:
DRAGON MAZE cont.

1255 XR=3*XR-2;YR=3*Y-2
1320 BY= RND (12)+1
1340 COLOR=6;VLIN 3*W-2,3*W+1
AT 35
1370 SX=13*Y-64
1380 GX=3*WX+2;GY=3*SY-2
1370 RD=1
1500 K=PEEK (-15064); IF K<168 THEN
1500
1510 JUKE -16868,6
1515 VX=K;GOSUB 7000;K=0;
1520 IF SX=K AND SY=K THEN 3000
1520 IF K= ASC('R') THEN 2000
1520 IF K= ASC('L') THEN 2500
1520 IF K= ASC('U') THEN 3000
1520 IF K= ASC('D') THEN 3500
1520 GOSUB 30000;GOTO 1500
2000 VX=100;VY=0
2010 IF VX+13*VX-1) MOD 16 THEN
2000
3000 GOTO 2000
4000 GOSUB 5000
4010 COLOR=15
4020 VLIN 3*SY-1,3*SY+1 AT 3*Y
4030 GOTO 1500
4100 GOSUB 5000
4100 COLOR=15
4120 VLIN 3*SY-1,3*SY+1 AT 3*Y
4130 GOTO 1500
4200 GOSUB 5000
4200 COLOR=15
4220 VLIN 3*SY-1,3*SY+1 AT 3*Y
4230 GOTO 1500
5000 S=5+1; FOR I=0 TO 288+2+PEEK
5000 (-16336)+PEEK (-16336)+PEEK (-16336)+PEEK (-16336)+PEEK
5000 I:RETURN
6000 PRINT "YOU WIN!"
6010 GOSUB 5000;GOSUB 50000;GOSUB 5000
6080 PRINT "SCORE=",S+3
6080 END
7000 IF VX>SX THEN 7005; IF VX>SY THEN
7000
7010 JUKE -16868,6
7020 GOSUB 5000
7020 COLOR=8
7030 PR=S+SY-2;RY=S+SY-2
7040 FOR I=1 TO 2;RX=RNY+RY;RY=RY+1
7040 COLOR=8
7050 FOR K=0 TO 1; FOR L=0 TO 1;
7050 FOR W=A;SY=1;NEXT L;K; COLOR=8
7050 NEXT R=0 TO 1; FOR L=0 TO 1;
7060 NEXT W
7070 NEXT I
7080 NEXT I
7080 NEXT I
7090 NEXT I
7090 NEXT I
7090 NEXT I
7090 NEXT I
7100 PRINT "YOU WIN!"
7110 GOSUB 5000;GOSUB 5000;GOSUB 5000;GOSUB 5000;
7100 END
DRAGON MAZE cont.

7110 DX=-1;DY=0: GOTO 7020
7150 IF SY=1 THEN 7005: IF T(SX+
13*(SY-1))>9 THEN 7160: IF
M(SX+13*(SY-1)-13)/10 THEN
7005
7160 DX=0;DY=-1: GOTO 7020
8000 GOSUB 5000: GOSUB 5000: GOSUB
5000: GOSUB 5000: PRINT 'THE DRA
GON GOT YOU!'
1999 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 CC (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 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;010.B410M</td>
<td>Copy the data now in the memory range from (adrs2) to (adrs3) into memory locations starting at (adrs1).</td>
</tr>
<tr>
<td>adrs3M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verify Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrs1&lt;adrs2</td>
<td>*100&lt;010.B410V</td>
<td>Verify that block of data in memory range from (adrs2) to (adrs3) exactly matches data block starting at memory location (adrs1) and displays 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>*3ØØ.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>*8ØØ.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.L</td>
<td>*C8ØØL</td>
<td>Decodes 2Ø instructions starting at memory (adrs) into 65Ø2 assembly mnemonic code.</td>
</tr>
<tr>
<td>L</td>
<td>*L</td>
<td>Decodes next 2Ø 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>$(monitor</td>
<td>!$C8ØØL</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>adrsl:(65Ø2</td>
<td>!CØ10:STA 23FF</td>
<td>Assembles a mnemonic 65Ø2 instruction into machine codes. If error, machine will refuse instruction, sound bell, and reprint line with up arrow under error.</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 mnemonic</td>
<td>! STA ØFF</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**

- **adrsG**
  - *3ØØG*
  - Runs machine level program starting at memory (adrs).

- **adrsT**
  - *8ØØT*
  - 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).

- **asrdS**
  - *CØ5ØS*
  - 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).

- **(Control E)**
  - *E C*
  - Displays 65Ø2 status registers and opens them for modification (see Note 1).

- **(Control Y)**
  - *Y C*
  - Executes user specified machine language subroutine starting at memory location (3F8).

**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.

**Hexadecimal Arithmetic**

- **datal+data2**
  - *78+34*
  - Performs hexadecimal sum of datal plus data2.

- **datal-data2**
  - *AE-34*
  - Performs hexadecimal difference of datal minus data2.
<table>
<thead>
<tr>
<th>Command Format</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*100L 400G AFFT</td>
<td>Multiple monitor commands may be given on same line if separated by a &quot;space&quot;.</td>
</tr>
<tr>
<td>*LLLL</td>
<td>Single letter commands may be repeated without spaces.</td>
</tr>
</tbody>
</table>
SPECIAL CONTROL AND EDITING CHARACTERS

"Control" characters are indicated by a super-scripted "C" such as G<sup>C</sup>. They are obtained by holding down the CTRL key while typing the specified letter. Control characters are NOT displayed on the TV screen. B<sup>C</sup> and C<sup>C</sup> must be followed by a carriage return. Screen editing characters are indicated by a sub-scripted "E" such as D<sub>E</sub>. 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<sup>C</sup> moves to cursor to right and copies text while A<sub>E</sub> 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 functions 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&lt;sup&gt;C&lt;/sup&gt;. 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.
SPECIAL CONTROL AND EDITING CHARACTERS
(continued)

<table>
<thead>
<tr>
<th>CHARACTER</th>
<th>DESCRIPTION OF ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_E$</td>
<td>Move cursor to right</td>
</tr>
<tr>
<td>$B_E$</td>
<td>Move cursor to left</td>
</tr>
<tr>
<td>$C_E$</td>
<td>Move cursor down</td>
</tr>
<tr>
<td>$D_E$</td>
<td>Move cursor up</td>
</tr>
<tr>
<td>$E_E$</td>
<td>Clear text from cursor to end of line</td>
</tr>
<tr>
<td>$F_E$</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></td>
<td><strong>Display Mode Controls</strong></td>
<td></td>
</tr>
<tr>
<td>CO50</td>
<td>10 POKE -16304,0</td>
<td>Set color graphics mode</td>
</tr>
<tr>
<td>CO51</td>
<td>20 POKE -16303,0</td>
<td>Set text mode</td>
</tr>
<tr>
<td>CO52</td>
<td>30 POKE -16302,0</td>
<td>Clear mixed graphics</td>
</tr>
<tr>
<td>CO53</td>
<td>40 POKE -16301,0</td>
<td>Set mixed graphics (4 lines text)</td>
</tr>
<tr>
<td>CO54</td>
<td>50 POKE -16300,0</td>
<td>Clear display Page 2 (BASIC commands use Page 1 only)</td>
</tr>
<tr>
<td>CO55</td>
<td>60 POKE -16299,0</td>
<td>Set display to Page 2 (alternate)</td>
</tr>
<tr>
<td>CO56</td>
<td>70 POKE -16298,0</td>
<td>Clear HIRES graphics mode</td>
</tr>
<tr>
<td>CO57</td>
<td>80 POKE -16297,0</td>
<td>Set HIRES graphics mode</td>
</tr>
<tr>
<td></td>
<td><strong>TEXT Mode Controls</strong></td>
<td></td>
</tr>
<tr>
<td>0020</td>
<td>90 POKE 32,L1</td>
<td>Set left side of scrolling window to location specified by L1 in range of 0 to 39.</td>
</tr>
<tr>
<td>0021</td>
<td>100 POKE 33,W1</td>
<td>Set window width to amount specified by W1. L1+W1&lt;40. W1&gt;0</td>
</tr>
<tr>
<td>0022</td>
<td>110 POKE 34,T1</td>
<td>Set window top to line specified by T1 in range of 0 to 23</td>
</tr>
<tr>
<td>0023</td>
<td>120 POKE 35,B1</td>
<td>Set window bottom to line specified by B1 in the range of 0 to 23. B1&gt;T1</td>
</tr>
<tr>
<td>0024</td>
<td>130 CH=PEEK(36)</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>140 POKE 36,CH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>150 TAB(CH+1)</td>
<td></td>
</tr>
<tr>
<td>0025</td>
<td>160 CV=PEEK(37)</td>
<td>Similar to above. Read/set cursor vertical position in the range 0 to 23.</td>
</tr>
<tr>
<td></td>
<td>170 POKE 37,CV</td>
<td></td>
</tr>
<tr>
<td></td>
<td>180 VTAB(CV+1)</td>
<td></td>
</tr>
<tr>
<td>0032</td>
<td>190 POKE 50,127</td>
<td>Set inverse flag if 127 (Ex. 190)</td>
</tr>
<tr>
<td></td>
<td>200 POKE 50,255</td>
<td>Set normal flag if 255(Ex. 200)</td>
</tr>
<tr>
<td>FC58</td>
<td>210 CALL -936</td>
<td>(@E) Home cursor, clear screen</td>
</tr>
<tr>
<td>FC42</td>
<td>220 CALL -958</td>
<td>(FE) Clear from cursor to end of page</td>
</tr>
<tr>
<td>Hex</td>
<td>BASIC Example</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>FC9C</td>
<td>23Ø CALL -868</td>
<td>(Eɕ) Clear from cursor to end of line</td>
</tr>
<tr>
<td>FC66</td>
<td>24Ø CALL -922</td>
<td>(JȻ) Line feed</td>
</tr>
<tr>
<td>FC70</td>
<td>25Ø CALL -912</td>
<td>Scroll up text one line</td>
</tr>
</tbody>
</table>

**Miscellaneous**

| C030 | 360 X=PEEK(-16336) | Toggle speaker |
| 365 POKE -16336,0 |
| C00Ø | 370 X=PEEK(-16384) | Read keyboard; if X>127 then key was pressed. |
| C010 | 380 POKE -16368,0 | Clear keyboard strobe - always after reading keyboard. |
| C061 | 390 X=PEEK(16287) | Read PDL(0) push button switch. If X>127 then switch is "on". |
| C062 | 400 X=PEEK(-16286) | Read PDL(1) push button switch. |
| C063 | 410 X=PEEK(-16285) | Read PDL(2) push button switch. |
| C058 | 420 POKE -16296,0 | Clear Game I/O A0 output |
| C059 | 430 POKE -16295,0 | Set Game I/O A0 output |
| C05A | 440 POKE -16294,0 | Clear Game I/O A1 output |
| C05B | 450 POKE -16293,0 | Set Game I/O A1 output |
| C05C | 460 POKE -16292,0 | Clear Game I/O A2 output |
| C05D | 470 POKE -16291,0 | Set Game I/O A2 output |
| C05E | 480 POKE -16290,0 | Clear Game I/O A3 output |
| C05F | 490 POKE -16289,0 | Set Game I/O A3 output |
TITLE "APPLE II SYSTEM MONITOR"
LOC0 EPZ $00
LOC1 EPZ $01
WNDLFT EPZ $20
WNDWIDTH EPZ $21
WNDTOP EPZ $22
WDBTM EPZ $23
CH EPZ $24
CV EPZ $25
GBASL EPZ $26
GBASH EPZ $27
BASL EPZ $28
BASH EPZ $29
BAS2L EPZ $2A
BAS2H EPZ $2B
H2 EPZ $2C
LMNEM EPZ $2C
RTNL EPZ $2C
V2 EPZ $2D
RNEMEP EPZ $2D
RTNH EPZ $2D
MASK EPZ $2E
CHKSUM EPZ $2E
FORMAT EPZ $2E
LASTIN EPZ $2F
LENGTH EPZ $2F
SIGN EPZ $2F
COLOR EPZ $30
MODE EPZ $31
INVFLG EPZ $32
PROMPT EPZ $33
YSAV EPZ $34
YSAV1 EPZ $35
CSWL EPZ $36
CSWH EPZ $37
KSWL EPZ $38
KSWH EPZ $39
PCL EPZ $3A
PCH EPZ $3B
XQT EPZ $3C
A1L EPZ $3C
A1H EPZ $3D
A2L EPZ $3E
A2H EPZ $3F
A3L EPZ $40
A3H EPZ $41
A4L EPZ $42
A4H EPZ $43
A5L EPZ $44
A5H EPZ $45
EQU ACC $45
EQU XREG $46
EQU YREG $47
EQU STATUS $48
EQU SYNT $49
EQU ENDL $4A
EQU RNDH $4F
EQU ACL $50
EQU ACH $51
EQU XTNDL $52
EQU XTNDH $53
EQU AUXL $54
EQU AUXH $55
EQU IN $200
EQU IOADR $0000
EQU KBD $C000
EQU KBDSTRB $C010
EQU TAPEOUT $C020
EQU SPKR $C030
EQU TXTCLR $C050
EQU TXTSET $C051
EQU MIXCLR $C052
EQU MIXSET $C053
EQU LOWSCR $C054
EQU HISCR $C055
EQU LORES $C056
EQU HIRES $C057
EQU TAPEIN $C060
EQU PADDLO $C064
EQU PTRIG $C070
EQU BASIC $E000
EQU BASIC2 $E003

ORG $F800 ROM START ADDRESS

F800: 4A PLOT LSR Y-COORD/2
F801: 08 PHP SAVE LSB IN CARRY
F802: 20 47 48 JSR GBASCALC CALC BASE ADR IN GBASL,H
F805: 28 PLP RESTORE LSB FROM CARRY
F806: A9 0F LDA #$0F MASK $0F IF EVEN
F808: 90 02 BCC RTMASK XOR COLOR
F80A: 69 E0 ADC #$E0 MASK $F0 IF ODD
F80C: 85 2E RTMASK STA MASK

F80B: B1 26 PLOT1 LDA (GBASL),Y DATA
F810: 45 30 EOR COLOR XOR DATA
F812: 25 2E AND MASK AND MASK
F814: 51 26 EOR (GBASL),Y XOR DATA

F816: 90 F6 BCC HLINE1 ALWAYS TAKEN

F818: 69 01 VLINEZ ADC #$01 NEXT Y-COORD
F819: 48 VLINE PHA SAVE ON STACK
F81A: 20 00 F8 JSR PLOT PLOT SQUARE
F81C: 68 PLA
F81D: C5 2D CMP V2 DONE?
F81E: 90 05 BCC VLINEZ NO, LOOP

F831: 60 RTS1 RTS
F832: A0 2F CLRSRC LDA #$2F MAX Y, FULL SCRn CLR
F834: D0 02 BNE CLRSRC2 ALWAYS TAKEN
F836: A0 27 CLETOP LDA #$27 MAX Y, TOP SCREEN CLR

F838: 84 2D CLRSRC2 STY V2 STORE AS BOTTOM COORD

F83A: A0 27 LDA #$27 RIGHTMOST X-COORD (COLUMN)

F83C: A0 00 CLRSRC3 LDA #$00 TOP COORD FOR VLINE CALLS

F83E: 85 30 STA COLOR CLEAR COLOR (BLACK)
F840: 20 28 F8 JSR VLINE DRAW VLINE
F844: 88 DEY NEXT LEFTMOST X-COORD
F844: 10 F6 BPL CLRSRC LOOP UNTIL DONE

F846: 60 RTS

F847: 48 GBASCALC PHA FOR INPUT 000DEFGH
F848: 4A LSR
F849: 29 03 AND #$03
F84B: 09 04 ORA #$04 GENERATE GBASH=000001FG
F84D: 85 27 STA GBASH
F84F: 68 PLA AND GBASH=HDEDE000

F850: 29 18 AND #$18
F852: 90 02 BCC GBCALC
F854: 69 7F ADC #$7F
F856: 85 26 GBCALC STA GBASL
FA40: FF FF FF  DFB $FF,$FF,$FF
FA43: 20 D0 F8  STEP JSR INSTDSP DISASSEMBLE ONE INST
FA46: 68  PLA AT (PCL,H)
FA47: 85 2C STA RTNL ADJUST TO USER
FA49: 68  PLA STACK SAVE
FA4A: 85 2D STA RTNH RTN ADR.
FA4C: A2 08 LDX $508
FA4E: BD 10 F8 XQINIT LDA INITBL-1,X INIT XEQ AREA
FA51: 95 3C STA XQT,X
FA53: CA  DEX
FA54: D0 F8 BNE XQINIT
FA56: A1 3A LDA (PCL,X) USER OPCODE BYTE
FA57: F0 42 BEQ XBRK F0 42 BEQ XBRK SPECIAL IF BREAK
FA5A: A4 2F LDV LENGTH LEN FROM DISASSEMBLY
FA5C: C9 20 CMP $20
FA5E: F0 59 BEQ XJSR HANDLE JSR, RTS, JMP,
FA60: C9 60 CMP $60 JMP (), RTI SPECIAL
FA62: F0 45 BEQ XRTS
FA64: C9 4C CMP $4C
FA66: F0 5C BEQ XJMP
FA68: C9 6C CMP $6C
FA6A: F0 59 BEQ XJMPAT
FA6C: C9 40 CMP $40
FA6E: F0 35 BEQ XRTI
FA70: 29 1F AND $1F
FA72: 49 14 EOR #$14
FA74: C9 04 CMP #$04 COPY USER INST TO XEQ AREA
FA76: F0 02 BEQ XQ2 WITH TRAILING NOPS
FA77: B1 3A XQ1 LDA (PCL),Y CHANGE REL BRANCH
FA77: 99 3C 00 XQ2 STA XQT,Y DISP TO 4 FOR
FA7D: 88 DEY JMP TO BRANCH OR
FA7E: 10 F8 BPL XQ1 NBRANCH FROM XEQ.
FA80: 20 3F FF JSR RESTORE RESTORE USER REG CONTENTS.
FA83: 4C 3C 00 JMP XQT USER OP FROM RAM
FA86: 85 45 IRQ STA ACC (RETURN TO NBRANCH)
FA88: 68 PLA
FA89: 48 PHA **IRQ HANDLER
FA8A: 0A ASL A
FA8B: 0A ASL A
FA8C: 0A ASL A
FA8D: 30 03 BMI  BREAK TEST FOR BREAK
FA8E: 6C FE 03 JMP (IRQLOC) USER ROUTINE VECTOR IN RAM
FA92: 28 BREAK PLP
FA93: 20 4C FF JSR SAV1 SAVE REG’S CN BREAK
FA95: 68 PLA INCLUDING PC
FA97: 85 3A STA PCL
FA99: 68 PLA
FA9A: 85 3B STA PCH
FA9C: 20 82 F8 XBRK JSR INSTDSP1 PRINT USER PC.
FA9F: 20 DA FA JSR RGDSP1 AND REG’S
FAA1: 4C 65 FF JMP (IRQLOC) USER ROUTINE VECTOR IN RAM
FAA5: 18 XRTI CLC
FAA6: 68 PLA SIMULATE RTI BY EXPECTING
FAA7: 85 48 STA STATUS STATUS FROM STACK, THEN RTS
FAA9: 68 XRTS PLA RTS SIMULATION
FAAA: 85 3A STA PCL EXTRACT PC FROM STACK
FAAC: 68 PLA AND UPDATE PC BY 1 (LEN=0)
FAAD: 85 3B STA PCH
FAAF: 20 82 F8 XBRK JSR INSTDSP1 PRINT USER PC.
FAB1: 20 56 F9 JSR PCADJ3 AND REG’S
FAB2: 4C 65 FF JMP MON GO TO MONITOR
FAB5: 18 XRTI CLC
FAB6: 68 PLA SIMULATE RTI BY EXPECTING
FAB7: 85 48 STA STATUS STATUS FROM STACK, THEN RTS
FAB9: 68 XRTS PLA RTS SIMULATION
FABC: 85 3A STA PCL EXTRACT PC FROM STACK
FABD: 85 3B STA PCH
FABF: A5 2F PCINC3 LDA LENGTH UPDATE PC BY LEN
FAC1: 20 56 F9 JSR PCADJ3
FAC4: 18 XJMP CLC
FAC5: B1 3A XJMPAT LDA (PCL),Y LOAD PC FOR JMP,
FAC7: AA TAX (JMP) SIMULATE.
FAC8: 88 DEY
FAC9: B1 3A LDA (PCL),Y
FACC: 86 3B STX PCH
FACD: 85 3A NEWPCL STA PCL
FACF: B0 F3 BCS XJMP
FADD: A5 2D RTNJMP LDA RTNH
FADF: 48 PHA
FADG: A5 2C LDA RTNL
FADH: 48 PHA
FAD7: 20 8E FD RGDSP JSR CRDOUT DISPLAY USER REG
FADA: A9 45 RDGSP1 LDA #ACC CONTENTS WITH
FADC: 85 40 STA A3L LABELS
FCC0: E5 3F SBC A2H
FCC2: E6 3C INC A1L (CARRY SET IF &gt;=)
FCC4: D0 02 BNE RTS4B
FCC6: E6 3D INC A1H
FCC8: 60 RTS4B RTS
FCC9: A0 4B HEADR LDY #$4B WRITE A*256 'LONG 1'
FCCB: 20 DB FC JSR ZERDLY HALF CYCLES
FCCD: D0 32 BNE HEADR (650 USEC EACH)
FCE0: 40 RTS4B RTS
FCE2: 48 DB FC WRBIT LDY #$4B WRITE A*256 'LONG 1'
FCE4: A0 32 LDY #$32 TIMING LOOP
FCE6: 60 RTS
FCE8: A2 08 RDBYTE LDX #$08 8 BITS TO READ
FCEA: A5 32 NOTCR LDA INVFLG ECHO USER LINE
FCEC: A2 08 RDBYTE PHA READ TWO TRANSITIONS
FCED: 20 PA FC JSR RD2BIT (FIND EDGE)
FCEF: 20 PA FC JSR RD2BIT (FIND EDGE)
FCF0: D0 02 BNE RTS4B
FCF1: 00 RTS
FCF2: 68 PLA
FCF3: 2A ROL NEXT BIT
FCF4: A0 3A LDY #$3A COUNT FOR SAMPLES
FCF5: CA DEX
FCF6: 60 RTS
FCF8: 88 RDBIT DEY DECR Y UNTIL
FCFB: A0 2C LDY #$2C
FCFD: 88 RDBIT DEY DECR Y UNTIL
FCFE: AD 60 CO LDA TAPEIN TAPE TRANSITION
FCF0: 45 2F EOR LASTIN INC RNDL
FCF1: D0 02 BNE KEYIN2 INCR RND NUMBER
FCF2: E6 4F INC RNDH
FCF3: 2C 00 C0 KEYIN2 BIT KBD KEY DOWN?
FCF4: 10 F5 BPL KEYIN LOOP
FCF5: 91 28 STA (BASL),Y REPLACE FLASHING SCREEN
FCF6: 09 40 ORA #$40
FCF7: 91 28 STA (BASL),Y
FCF8: 68 PLA
FCF9: 6C 38 00 JMP ($XSL) GO TO USER KEY-IN
FCFA: E6 4E KEYIN INC RNDL
FCFB: D0 02 BNE KEYIN2 INCR RND NUMBER
FCFC: E6 4F INC RNDH
FCFD: 2C 00 C0 KEYIN2 BIT KBD KEY DOWN?
FCFE: 10 F5 BPL KEYIN LOOP
FCFF: 91 28 STA (BASL),Y REPLACE FLASHING SCREEN
F000: AD 60 CO LDA KBD GET KEYCODE
F002: 2C 10 CO BIT KBSTRB CLR KEY STROBE
F004: 60 RTS
F006: 20 0C FD ESC JSR RDBIT GET KEYCODE
F008: 20 0C FD ESC JSR RDBIT GET KEYCODE
F00A: 20 0C FD ESC JSR RDBIT GET KEYCODE
F00C: 20 0C FD ESC JSR RDBIT GET KEYCODE
F00E: 20 0C FD ESC JSR RDBIT GET KEYCODE
F010: 2C 00 C0 KEYIN2 BIT KBD KEY DOWN?
F011: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F012: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F013: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F014: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F015: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F016: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F017: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F018: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F019: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F01A: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F01B: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F01C: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F01D: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F01E: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F01F: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F020: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F021: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F022: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F023: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F024: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F025: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F026: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F027: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F028: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F029: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F02A: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F02B: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F02C: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F02D: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F02E: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F02F: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F030: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F031: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F032: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F033: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
F034: 2F 00 C0 KEYIN2 BIT KBD KEY DOWN?
FE0F: 91 40 STA (A3L),Y STORE AS LOW BYTE AS (A3)
FE11: E6 40 INC A3L
FE13: D0 02 BNE RTS5 INCR A3, RETURN
FE15: E6 41 INC A3H
FE17: 60 RTS5 RTS
FE18: A4 34 SETMODE LDY YSAV SAVE CONVERTED ':', '+',
FE1A: B9 FF 01 LDA IN-1,Y ' - ', ' - ' AS MODE.
FE1D: 85 31 SETMDX STA MODE
FE1F: 60 RTS
FE20: A2 01 LT LDX #$01
FE22: B5 3E LT2 LDA A2L,X COPY A2 (2 BYTES) TO
FE24: 95 42 STA A4L,X A4 AND A5
FE26: 95 44 STA A5L,X
FE28: CA DEX
FE29: 10 F7 JSR NXTA4
FE2B: 60 RTS
FE2C: B1 3C MOVE LDA (A1L),Y MOVE (A1 TO A2) TO
FE2E: 91 42 STA (A4L),Y (A4)
FE30: 20 B4 FC JSR NXTA4
FE33: 90 F7 BCC MOVE
FE35: 60 RTS
FE36: B1 3C VFY LDA (A1L),Y VERIFY (A1 TO A2) WITH
FE38: D1 42 CMP (A4L),Y (A4)
FE3A: F0 1C BEQ VFYOK
FE3C: 20 92 FD JSR PRA1
FE3E: B9 FF 01 LDA IN-1,Y AS MODE.
FE40: A4 34 SETMODE LDY YSAV SAVE CONVERTED ':', '+',
FE42: B9 FF 01 LDA IN-1,Y ' - ', ' - ' AS MODE.
FE44: 85 31 SETMDX STA MODE
FE46: 60 RTS
FE48: B1 3C MOVE LDA (A1L),Y MOVE (A1 TO A2) TO
FE4A: 91 42 STA (A4L),Y (A4)
FE4C: 20 B4 FC JSR NXTA4
FE4E: 60 RTS
FE50: 20 75 FE LIST JSR A1PC MOVE A1 (2 BYTES) TO
FE52: A9 14 LDA #$14 PC IF SPEC'D AND
FE54: 48 LIST2 PHA DISEMBLE 20 INSTRS
FE56: 20 D0 F8 JSR INSTDSP
FE58: 20 53 F9 JSR PCDATA ADJUST PC EACH INSTR
FE5A: 85 3A STA PCL
FE5C: 84 3B STT PCH
FE5E: 68 PLA
FE60: 38 SEC
FE62: E9 01 SBC #$01 NEXT OF 20 INSTRS
FE64: D0 0F BNE LIST2
FE66: 60 RTS
FE68: 8A A1PC TXA IF USER SPEC'D ADDR
FE6A: F0 07 BEQ A1PCRTS COPY FROM A1 TO PC
FE6C: B5 3C A1PCRTS LDA A1L,X
FE70: 95 3A STA PCL,X
FE72: CA DEX
FE74: 10 F9 JSR A1PCRTS
FE76: 60 A1PCRTS RTS
FE78: A0 3F SETINV LDY #$3F SET FOR INVERSE VID
FE7A: A9 00 SETKBD LDA #$00 SIMULATE PORT #0 INPUT
FE7C: 85 3E IMPORT STA A2L SPECIFIED (KEYIN ROUTINE)
FE7E: A2 38 INPRT LDX #$KML
FE80: A0 1B LDA #$KEYIN
FE82: D0 08 BNE IOPRT
FE84: A9 00 SERTKBD LDA #$50 SIMULATE PORT #0 INPUT
FE86: 85 3E IMPORT STA A2L SPECIFIED (KEYIN ROUTINE)
FE88: A2 38 INPRT LDX #$KML
FE8A: A0 1B LDA #$KEYIN
FE8C: A9 00 SERTKBD LDA #$50 SIMULATE PORT #0 INPUT
FE8E: A2 38 INPRT LDX #$KML
FE90: A0 1B LDA #$KEYIN
FE92: D0 02 BNE IOPRT
FE94: A9 00 SERTKBD LDA #$50 SIMULATE PORT #0 OUTPUT
FE96: 85 3E IMPORT STA A2L SPECIFIED (COUT1 ROUTINE)
FE98: A2 38 INPRT LDX #$KML
FE9A: A0 F0 LDA #$COUT1
FE9C: A5 3E IOPRT LDA A2L SRT RAM IN/OUT VECTORS
FE9E: 29 0F AND #$0F
FEA0: E0 06 BNE IOPRT1
FEA2: A0 00 LDY #$00
FEA4: F0 02 BNE IOPRT2
FEA6: A9 00 IOPRT1 LDA #$COUT1/256
FEA8: 94 00 IOPRT2 STA LOC0,X
FEAA: 95 01 STA LOC1,X
FEAC: 60 RTS
FEAD: 8A EA NOP
FEAE: 4C 00 JMP BASIC TO BASIC WITH SCRATCH
FEF0: 4C 03 JMP BASIC2 CONTINUE BASIC
FEB6: 20 75 FE GO JSR A1PC ADR TO PC IF SPEC'D
FEB9: 20 3F FF JSR RESTORE RESTORE META REGS
FEBD: 20 4C D7 FA REGP JMP REGP TO REG DISPLAY
FEC2: C6 34 TRACE DEC YSAY
FEC4: 20 75 FE STEPZ JSR A1PC ADR TO PC IF SPEC'D
FEC7: 4C 43 FA JMP STEP TAKE ONE STEP
FEC8: 4C F8 03 USR JMP USRADDR TO USER SUBR AT USRADDR
FECF: A9 40 WRITE LDA #$40
FECF: 20 C9 FC JSR HEADR WRITE 10-SEC HEADER
FED2: A0 27 LDY #$27
FED4: A0 00 LDY #$00
FED6: 41 3C EOR (A1L,X)
FED8: 48 PLA
FED9: A1 3C REGZ JMP REGDSP TO REG DISPLAY
FEDC: A9 1D LDY #$1D
FEED: 68 PLA
FEE1: 20 ED FE JSR WRBYTE
FEE4: 20 00 FE CRMON JSR BL1 HANDLE A CR AS BLANK
FEE7: 68 PLA
FEE8: 68 PLA
FEEA: 68 PLA AND RTN TO MON
FEEB: D0 6C BNE MONZ
FEEC: 20 FA FC JSR RD2BIT FIND TAPEIN EDGE
FEEF: A9 16 LDA #$16
FFF0: A9 C5 JSR RD2BIT FIND TAPEIN EDGE
F000: A9 45 JSR RD2BIT (SHORT 0)
F002: 20 09 BCS RD2 LOOP UNTIL FOUND
F007: 20 00 FA JSR RD2B1 SKIP SECOND SYNC H-CYCLE
F00A: A0 44 RD2 LDY #$44 INDEX FOR 0/1 TEST
F00D: 20 EC FC JSR RDBYTE READ A BYTE
F010: 85 2E STA CHKSUM INIT CHKSUM=$FF
F013: 20 FD FC JSR RDBYTE READ CHKSUM BYTE
F016: C5 2E CMP CHKSUM
F019: F0 0D BNE BELL GOOD, SOUND BELL AND RETURN
F01C: A9 C5 PRERR LDA #$C5
F020: 20 ED FD JSR COUT PRINT "ERR", THEN BELL
F023: A9 D2 LDA #$D2
F026: 20 3A FF JSR BELL
F029: A9 87 BELL LDA #$87 OUTPUT BELL AND RETURN
F02C: 4C 48 JMP COUT
F02F: A5 48 RESTORE LDA STATUS RESTORE 6502 REG CONTENTS
F032: 20 00 FE JSR SETNORM SET SCREEN MODE
F035: 20 2F FA JSR INIT AND INIT KEI/SCREEN
F038: A9 9A LDA #$9A
F03B: 08 PHP
F03C: 68 PLA
F03D: 85 48 STA STATUS
F03E: BA TIX
F041: 86 49 STX SPNT
F042: 20 84 FE JSR GETMN READ A LINE
FFF0: 83  DFB  SETNORM-1
FFF1: 7F  DFB  SETINV-1
FFF2: 5D  DFB  LIST-1
FFF3: CC  DFB  WRITE-1
FFF4: B5  DFB  GO-1
FFF5: FC  DFB  READ-1
FFF6: 17  DFB  SETMODE-1
FFF7: 17  DFB  SETMODE-1
FFF8: F5  DFB  CRMON-1
FFF9: 03  DFB  BLANK-1
FFFA: FB  DFB  NMI   NMI VECTOR
FFFF: 03  DFB  NMI/256
FFFF: 59  DFB  RESET  RESET VECTOR
FFFF: FF  DFB  RESET/256
FFFF: 86  DFB  IRQ    IRQ VECTOR
FFFF: FA  DFB  IRQ/256
XQTNZ  EQU  $3C
TITLE "APPLE-II MINI-ASSEMBLER"

ORG $F500

F500: E9 81 REL SBC #81 IS FMT COMPATIBLE
F502: 4A LSR WITH RELATIVE MODE?
F503: D0 14 BNE ERR3 NO.
F505: A4 3F LDY A2H
F507: A6 3E LDX A2L DOUBLE DECREMENT
F509: D0 01 BNE REL2
F50B: 88 DEY
F50C: CA REL2 DEX
F50D: 8A TXA
F50E: 18 CLC
F50F: E5 3A SBC PCL FORM ADDR-PC-2
F511: 85 3E STA A2L
F513: 10 01 BPL REL3
F515: C8 INY
F516: 98 REL3 TYA
F517: E5 3B SBC PCH
F519: D0 6B ERR3 BNE ERR ERROR IF >1-BYTE BRANCH
F51B: A4 2F FINDOP LDY LENGTH
F51D: B9 3D 00 FNDOP2 LDA A1H,Y MOVE INST TO (PC)
F520: 91 3A DEY
F522: 88 FPL FNDOP2
F525: 20 1A FC JSR CURSUP
F528: 20 1A FC JSR INSTDSP TYPE FORMATTED LINE
F52B: 20 D0 F8 JSR INSTDSP TYPE FORMATTED LINE
F52E: 20 53 F9 JSR PCADJ UPDATE PC
F531: 84 3B STY PCH
F533: 85 3A STA PCL
F535: 4C 95 F5 JMP NXTLINE GET NEXT LINE
F538: 20 BE FF FAKEMON3 JSR TOSUB GO TO DELIM HANDLER
F53B: A4 34 LDY YSAV RESTORE Y-INDEX
F53D: 20 A7 FF FAKEMON JSR GETNUM READ PARAM
F540: 84 34 STY YSAV SAVE Y-INDEX
F542: A5 31 LDA MODE
F544: A0 00 LDY #$0 INIT DELIMITER INDEX
F546: 88 DEY CHECK NEXT DELIM
F548: D9 CC FF CMP CHRTBL,Y COMPARE WITH DELIM TABLE
F54A: D0 F8 BNE FAKEMON2 NO MATCH
F54C: C0 15 CPY #$15 MATCH, IS IT CR?
F54E: A5 31 LDA A1H GET TRIAL OPCODE
F550: A0 00 LDY #$0
F552: A5 3D TRYNEXT LDA A1H GET TRIAL OPCODE
F554: 98 ERR2 TYA
F556: A9 DE LDA #$DE CHAR TO INDICATE ERROR
F558: C9 93 CMP #$93 COL 1 TERMINATOR?
F55A: AD 00 02 LDA IN CHAR
F55C: C9 A0 CMP #$A0 ASCII BLANK?
F55E: A9 03 SPACE LDA #$3 COUNT OF CHARS IN MNEMONIC
F55F: 85 34 06 NXTMN JSR GSPNSP GET FIRST MNEM CHAR.
F560: FA A0 NXTM ASL A
F561: E9 BE SBC #$BE SUBTRACT OFFSET
F562: C9 03 SPACE LDA #$3 COUNT OF CHARS IN MNEMONIC
F563: 20 34 P6 NXTMN JSR GETMN CHAR
F564: 0A NXTM ASL A
F565: 20 4A P9 JSR PRL2 PRINT ^ UNDER LAST READ
F567: D0 13 BNE NEXTOP NO, TRY NEXT OPCODE
F569: BD C0 F9 LDA MNEML,X GET UPPER MNEMONIC BYTE
F56C: C5 43 CMP A4H MATCH?
F570: A5 44 LDA FMT GET TRIAL FORMAT
F572: A4 2E LDY FORMAT GET TRIAL FORMAT
F574: C0 9D CPY #$9D TRIAL FORMAT RELATIVE?
F576: F0 9F BEQ FINDOP YES.
F578: C6 3D NEXTOP - DEC A1H NO, TRY NEXT OPCODE
F57C: C6 3D NEXTOP DEC A1H NO, TRY NEXT OPCODE
F57E: D0 DC LDA TRYNEXT
F580: E6 44 INC FMT NO MORE, TRY WITH LEN=2
F582: C6 35 DEC L WAS L=2 ALREADY?
F584: F0 D6 BEQ TRYNEXT NO.
F586: A4 34 ERR LDY YSAV YES, UNRECOGNIZED INST.
F588: 98 ERR2 TAX
F58A: 20 4A P9 JSR PRL2 PRINT ^ UNDER LAST READ
F58D: A9 DE LDA #$DE CHAR TO INDICATE ERROR
F592: 20 3A FF RESETZ JSR BELL "L"
F593: 85 33 STA PROMPT INITIALIZE PROMPT
F595: A9 A1 LDA <$A1 "!
F597: 85 33 STA PROMPT INITIALIZE PROMPT
F599: 20 67 FD JSR GETLNZ GET LINE.
F59C: 20 C7 FF JSR ZMODE INIT SCREEN STUFF
F59F: AD 00 02 LDA IN GET CHAR
F5A2: C9 A0 CMP #$A0 ASCII BLANK?
F5A4: F0 13 BEQ SPACE YES
F5A6: C8 INY
F5A7: C9 A4 CMP #$A4 ASCII 'S' IN COL 1?
F5A9: F0 92 BEQ FAKEMON YES, SIMULATE MONITOR
F5AB: 88 DEY NO, BACKUP A CHAR
F5AC: 20 A7 FF JSR GETNUM GET A NUMBER
F5AF: C9 93 CMP #$93 '+' TERMINATOR?
F5B0: D1 D5 ERR4 BNE ERR2 NO, ERR.
F5B3: 8A TAX
F5B4: F0 D2 BEQ ERR2 NO ADR PRECEDING COLON.
F5B6: 20 78 FR JSR A1PCLP MOVE ADR TO PCL, PCH.
F5B9: A9 03 SPACE LDA #$3 COUNT OF CHARS IN MNEMONIC
F5BB: 85 3D STA A1H
F5BD: 20 34 P6 NXTMN JSR GSPNSP GET FIRST MNEM CHAR.
F5C0: 0A NXTM ASL A
F5C1: E9 BE SBC #$BE SUBTRACT OFFSET
F5C3: C9 C2 CMP #$C2 LEGAL CHAR?
F5C5: 90 C1 BCC ERR2 NO.
F5C7: 0A ASL A COMPRESS-LEFT JUSTIFY
F5C8: 0A ASL A
F5C9: A2 04 LDX #$4
F5CB: 0A NXTM ASL A DO 5 TRIPLE WORD SHIFTS
F5CC: 26 42   ROL A4L
F5CE: 26 43   ROL A4K
F5D0: CA   DEX
F5D1: 10 P8   BPL NXTM2
F5D3: C6 3D   DEC A1H   DONE WITH 3 CHARs?
F5D5: F0 F4   BEQ NXTM2   YES, BUT DO 1 MORE SHIFT
F5D7: 10 E4   BPL NXTMN   NO
F5D9: A2 05 FORM1 LDX #$5   5 CHARs IN ADDR MODE
F5DB: 20 34 P6 FORM2 JSR GETNSP   GET FIRST CHAR OF ADDR
F5DE: 84 34   STY YSAV
F5E0: DD B4 F9   CMP CHAR1,X   FIRST CHAR MATCH PATTERN?
F5E3: D0 13   BNE FORM3   NO
F5E5: 20 34 P6 JSR GETNSP   YES, GET SECOND CHAR
F5E8: DD BA F9   CMP CHAR2,X   MATCHES SECOND HALF?
F5EB: F0 OD   BEQ FORM5   YES.
F5ED: BD BA F9   LDA CHAR2,X   NO, IS SECOND HALF ZERO?
F5F0: F0 OD   BEQ FORM4   YES.
F5F2: C9 A4   CMP #$A4   NO, SECOND HALF OPTIONAL?
F5F4: F0 03   BEQ FORM4   YES.
F5F6: A4 34   ROL FMT   FORM FORMAT BYTE
F5F8: 18 FORM3 CLC   CLEAR BIT-NO MATCH
F5F9: 88 FORM4 DEY   BACK UP 1 CHAR
F5FA: 26 44 FORM5 ROL PMT   FORM FORMAT BYTE
F5FC: E0 03  CFX #$3   TIME TO CHECK FOR ADDR.
F5FE: D0 OD   BNE FORM7   NO
F600: 20 A7 FF   JSR GETNUM   YES
F603: A5 3F   LDX A2H
F605: F0 01   BEQ FORM6   HIGH-ORDER BYTE ZERO
F607: E8   INX   NO, INCR FOR 2-BYTE
F608: 86 35 FORM6 STX L   STORE LENGTH
F60A: A2 03   LDX #$3   RELOAD FORMAT INDEX
F60C: 88   DEY   BACK UP 1 CHAR
F60E: 86 3D FORM7 STX A1H   SAVE INDEX
F610: CA   DEX   DONE WITH FORMAT CHECK?
F612: 10 C9   BPL FORM2   NO.
F614: A5 44   LDA PMT   YES, PUT LENGTH
F616: 0A   ASL A   IN LOW BITS
F618: 0A   ASL A
F61A: 05 35 ORA L
F61C: C9 20   CMP #$20
F61E: B0 06   BCS FORM8   ADD "$" IF NONZERO LENGTH
F620: A6 35   LDX L   AND DON'T ALREADY HAVE IT
F622: A6 35   LDX #$3
F624: 85 35 FORM8 STA PMT
F626: 89 00 02   LDA IN,Y   GET NEXT NONBLANK
F628: 89 BB   LDA IN,X   GET NEXT NONBLANK
F62A: C9 BB   CMP #$BB   "" START OF COMMENT?
F62C: 80 04   BEQ FORM8   YES
F62E: 10 C9   BPL FORM9   YES
F630: 9D 8D   CMP #$8D   CARRIAGE RETURN?
F632: D0 80   BNE ERR4   NO, ERR.
F634: 4C 5C F5 FORM9 JMP TRYNEXT
F636: B9 00 02   GETNSP LDA IN,Y
F638: C8   INY
F63A: C9 A0   CMP #$A0   GET NEXT NON BLANK CHAR
F63C: F0 F8   BEQ GETNSP
F63E: 60   RTS
F640: ORG $F666
F642: 4C 92 F5 MINIASM JMP RESETZ
***************
*  APPLE-II FLOATING  *
*   POINT ROUTINES  *
*                     *
* COPYRIGHT 1977 BY  *
*  APPLE COMPUTER INC.  *
*                     *
* ALL RIGHTS RESERVED  *
*                     *
* S. WOZNIAK  *
*                     *
***************

TITLE "FLOATING POINT ROUTINES"
SIGN  EPZ  $F3
X2  EPZ  $F4
M2  EPZ  $F5
X1  EPZ  $F8
M1  EPZ  $F9
E  EPZ  $FC
OVLOC  EQU  $3F5
ORG  $F425

F425: 18  ADD  CLC  CLEAR CARRY
F426: A2 02  LDX  #$2  INDEX FOR 3-BYTE ADD.
F428: B5 F9  ADD1  LDA  M1,X
F42A: 75 F5  ADC  M2,X  ADD A BYTE OF MANT2 TO MANT1
F42C: 95 F9  STA  M1,X
F42E: CA  DEX  INDEX TO NEXT MORE SIGNIF. BYTE.
F42F: 10 F7  BPL  ADD1  LOOP UNTIL DONE.
F431: 60  RTS  RETURN
F432: 06 F3  MD1  ASL  SIGN  CLEAR LSB OF SIGN.
F434: 20 37 F4  JSR  ABSWAP  ABS VAL OF M1, THEN SWAP WITH M2
F437: 24 F9  ABSWAP  BIT  M1  MANT1 NEGATIVE?
F439: 10 05  BPL  ABSWAP1  NO, SWAP WITH MANT2 AND RETURN.
F43B: 20 A4 F4  JSR  FCOMPL  YES, COMPLEMENT IT.
F43E: E6 F3  INC  SIGN  INCR SIGN, COMPLEMENTING LSB.
F440: 38  ABSWAP1  SEC  SET CARRY FOR RETURN TO MUL/DIV.
F441: A2 04  SWAP  LDX  #$4  INDEX FOR 4 BYTE SWAP.
F443: 94 F8  SWAP1  STY  E-1,X
F445: B5 F7  LDA  X1-1,X  SNAP A BYTE OF EXP/MANT1 WITH
F447: B4 F3  LDY  X2-1,X  EXP/MANT2 AND LEAVE A COPY OF
F449: 94 F7  STY  X1-1,X  MANT1 IN E (3 BYTES). E+3 USED
F44B: 95 F3  STA  X2-1,X
F44D: CA  DEX  ADVANCE INDEX TO NEXT BYTE
F44E: D0 F3  BNE  SWAP1  LOOP UNTIL DONE.
F450: 60  RTS  RETURN
F451: A9 8E  FLOAT  LDA  #$8E  INIT EXP1 TO 14,
F453: 85 F8  STA  X1  THEN NORMALIZE TO FLOAT.
F455: A5 F9  NORM1  LDA  M1  HIGH-ORDER MANT1 BYTE.
F457: C9 C0  CMP  #$C0  UPPER TWO BITS UNEQUAL?
F459: 30 0C  BMI  RTS1  YES, RETURN WITH MANT1 NORMALIZED
F45B: C6 F8  DBC  X1  DECREMENT EXP1.
F45D: 06 F8  ASL  M1+2
F45F: 26 FA  ROL  M1  SHIFT MANT1 (3 BYTES) LEFT.
F461: 26 F9  ROL  M1
F463: A5 F8  NORM  LDA  X1  EXP1 ZERO?
F465: D0 BE  BNE  NORM1  NO, CONTINUE NORMALIZING.
F467: 60  RTS  RTS1  RETURN
F468: 20 A4 F4  FSUB  JSR  FCOMPL  CMPL MANT1, CLEARS CARRY UNLESS 0
F46B: 20 7B F4  SWPALGN  JSR  ALCONSMP  RIGHT SHIFT MANT1 OR SWAP WITH
F46E: A5 F4  FADD  LDA  X2
F470: C5 F8  CMP  X1  COMPARE EXP1 WITH EXP2.
F472: D0 F7  BNE  SWPALGN  IF #, SWAP ADDENDS OR ALIGN MANTS.
F474: 20 25 F4  JSR  ADD  ADD ALIGNED MANTISSAS.
F477: 50 BA  ADDEND  BVC  NORM  NO OVERFLOW, NORMALIZE RESULT.
F479: 70 05  BVS  RTLOG  CV: SHIFT M1 RIGHT, CARRY INTO SIGN
F47B: 90 C4 ALGNSWP BCC SWAP SWAP IF CARRY CLEAR, ELSE SHIFT RIGHT ARITH.
F47D: 0A RTAR LDA M1 SIGN OF MANT1 INTO CARRY FOR
F47F: 0A ASL RIGHT ARITH SHIFT.
F480: 02 RTLOG INC X1 INC X1 TO ADJUST FOR RIGHT SHIFT
F482: 75 BBQ OVFL EXP1 OUT OF RANGE.
F484: A2 FA RTLOG1 LDX #$FA INDEX FOR 6:BYTE RIGHT SHIFT.
F486: 76 7F ROTR ROR E+3,X
F488: E6 F3 RTAR LDA M1 SIGN OF MANT1 INTO CARRY FOR
F48A: 0E M1 INC X1 TO ADJUST FOR RIGHT SHIFT
F48C: 80 03 MUL2 IF CARRY CLEAR, SKIP PARTIAL PROD
F48E: 80 25 F4 JNZ ROR1 LOOP UNTIL DONE.
F490: 60 RTS RETURN.
F492: 20 32 F4 FMUL JSR MD1 ABS VAL OF MANT1, MANT2
F494: 65 F8 ADC X1 ADD EXP1 TO EXP2 FOR PRODUCT EXP
F496: A2 FA RTLOG1 LDX #$FA INDEX FOR 3:BYTE RIGHT SHIFT.
F498: 76 FF ROR1 ROR E+3,X
F49A: E8 INX NEXT BYTE OF SHIFT.
F49C: F0 75 BEQ OVFL EXP1 OUT OF RANGE.
F49E: A2 FA RTLOG1 LDX #$FA INDEX FOR 6:BYTE RIGHT SHIFT.
F4A0: 20 25 F4 JSR ADD ADD MULTIPLICAND TO PRODUCT.
F4A2: 60 RTS RETURN.
F4A4: 20 32 F4 FMUL JSR MD1 ABS VAL OF MANT1, MANT2
F4A6: 65 F8 ADC X1 ADD EXP1 TO EXP2 FOR PRODUCT EXP
F4A8: A2 FA RTLOG1 LDX #$FA INDEX FOR 3:BYTE RIGHT SHIFT.
F4AA: 76 FF ROR1 ROR E+3,X
F4AC: E8 INX NEXT BYTE OF SHIFT.
F4AD: 60 RTS RETURN.

Notes:
- The code appears to be part of a larger program, possibly for a specific processor or computing environment.
- The code includes various arithmetic and logical operations.
- The instructions are written in hexadecimal format with corresponding mnemonics.
- The code seems to be related to floating-point arithmetic, including multiplication, division, and floating-point operations such as sign determination and normalization.
- The presence of specific mnemonics like RTAR, RTLOG, RTLOG1, etc., indicates that the code is designed for a specific processor with a set of predefined instructions.
TITLE "SWEET16 INTERPRETER"
ORG $F689
F689: 20 4A FF  SW16     JSR   SAVE       PRESERVE 6502 REG CONTENTS
F68C: 68  PLA
F68D: 85 1K  STA   R15L       INIT SWEET16 PC
F68F: 68  PLA  FROM RETURN
F690: 85 1F  STA   R15H       ADDRESS
F692: 68  PLA
F693: 85 1E  STA   R14H       ONE SWEET16 INSTR.
F695: 4C 92  JMP   SW16B      INTERPRET AND EXECUTE
F697: E6 1E  INC   R15L
F699: D0 02  BNE   SW16D      ADDRESS
F69B: E6 1F  INC   R15H
F69D: A9 F7  LDA   #SW16PAG
F69F: 48  PHA  PUSH ON STACK FOR RTS
F6A0: 48  PHA
F6A1: A0 00  LDY   #$0
F6A4: 29 0F  AND   #$F    MASK SPECIFICATION
F6A6: AA  TAX
F6A7: 4A  LSR   A    OPCODE*2 TO LSB'S
F6A9: 4A  LSR   A
F6AA: 4A  LSR   A
F6AB: B9 E1 F6  LDA   OPTBL-2,Y  LOW ORDER ADR BYTE
F6B2: 48  PHA  ONTO STACK
F6B4: 60  RTS  GOTO REG-OP ROUTINE
F6B6: E6 1E  TOBR  INC   R15L
F6B8: D0 02  BNE   TOBR2  INCR PC
F6BA: E6 1F  INC   R15H
F6BC: BD E4 F6  TOBR2  LOW ORDER ADR BYTE
F6BE: 48  PHA  ONTO STACK FOR NON-REG OP
F6BF: A5 1D  LDA   R14H  'PRIOR RESULT REG' INDEX
F6C0: 4A  LSR   A  PREPARE CARRY FOR BC, BNC.
F6C1: 60  RTS  GOTO NON-REG OP ROUTINE
F6C3: 68  RTNZ PLA  POP RETURN ADDRESS
F6CE: 68  PLA
F6CF: 20 3F FF  JSR   RESTORE  RESTORE 6502 REG CONTENTS
F6D0: 6C 1E 00  JMP   (R15L) RETURN TO 6502 CODE VIA PC
F6D3: B1 1K  SETZ  LDA   (R15L),Y  HIGH-ORDER BYTE OF CONSTANT
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction 1</th>
<th>Instruction 2</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F7EB: 20 66 F7</td>
<td>JSR DCR</td>
<td>DCR STACK POINTER</td>
<td></td>
</tr>
<tr>
<td>F7EE: A1 00</td>
<td>LDA (R0L,X)</td>
<td>POP HIGH RETURN ADDRESS TO PC</td>
<td></td>
</tr>
<tr>
<td>F7F0: 85 1F</td>
<td>STA R15H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F7F2: 20 66 F7</td>
<td>JSR DCR</td>
<td>SAME FOR LOW-ORDER BYTE</td>
<td></td>
</tr>
<tr>
<td>F7F5: A1 00</td>
<td>LDA (R0L,X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F7F7: 85 1E</td>
<td>STA R15L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F7F9: 60</td>
<td>RTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F7FA: 4C C7 F6</td>
<td>RTN</td>
<td>JMP RTNZ</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ADC</strong></td>
<td>Add Memory to Accumulator with Carry</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>AND</strong></td>
<td>&quot;AND&quot; Memory with Accumulator</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ASL</strong></td>
<td>Shift Left One Bit (Memory or Accumulator)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BCC</strong></td>
<td>Branch on Carry Clear</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BCS</strong></td>
<td>Branch on Carry Set</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BEQ</strong></td>
<td>Branch on Result Zero</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BIT</strong></td>
<td>Test Bits in Memory with Accumulator</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BMI</strong></td>
<td>Branch on Result Minus</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BNE</strong></td>
<td>Branch on Result not Zero</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BPL</strong></td>
<td>Branch on Result Plus</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BRK</strong></td>
<td>Force Break</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BVC</strong></td>
<td>Branch on Overflow Clear</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BVS</strong></td>
<td>Branch on Overflow Set</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CLC</strong></td>
<td>Clear Carry Flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CLD</strong></td>
<td>Clear Decimal Mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CLI</strong></td>
<td>Clear Interrupt Disable Bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CLV</strong></td>
<td>Clear Overflow Flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CMP</strong></td>
<td>Compare Memory and Accumulator</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPX</strong></td>
<td>Compare Memory and Index X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPY</strong></td>
<td>Compare Memory and Index Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DEC</strong></td>
<td>Decrement Memory by One</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DEX</strong></td>
<td>Decrement Index X by One</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DEY</strong></td>
<td>Decrement Index Y by One</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EOR</strong></td>
<td>&quot;Exclusive-Or&quot; Memory with Accumulator</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>INC</strong></td>
<td>Increment Memory by One</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>INX</strong></td>
<td>Increment Index X by One</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>INY</strong></td>
<td>Increment Index 'I by One</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>JMP</strong></td>
<td>Jump to New Location</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>JSR</strong></td>
<td>Jump to New Location Saving Return Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LDA</strong></td>
<td>Load Accumulator with Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LDX</strong></td>
<td>Load Index X with Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LDY</strong></td>
<td>Load Index Y with Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LSR</strong></td>
<td>Shift Right one Bit (Memory or Accumulator)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NOP</strong></td>
<td>No Operation</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ORA</strong></td>
<td>OR Memory with Accumulator</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PHA</strong></td>
<td>Push Accumulator on Stack</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PHP</strong></td>
<td>Push Processor Status on Stack</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PLA</strong></td>
<td>Pull Accumulator from Stack</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PLP</strong></td>
<td>Pull Processor Status from Stack</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ROL</strong></td>
<td>Rotate One Bit Left (Memory or Accumulator)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ROR</strong></td>
<td>Rotate One Bit Right (Memory or Accumulator)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RTI</strong></td>
<td>Return from Interrupt</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RTS</strong></td>
<td>Return from Subroutine</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SBC</strong></td>
<td>Subtract Memory from Accumulator with Borrow</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SEC</strong></td>
<td>Set Carry Flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SED</strong></td>
<td>Set Decimal Mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SEI</strong></td>
<td>Set Interrupt Disable Status</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>STA</strong></td>
<td>Store Accumulator in Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>STX</strong></td>
<td>Store Index X in Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>STY</strong></td>
<td>Store Index Y in Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TAX</strong></td>
<td>Transfer Accumulator to Index X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TAY</strong></td>
<td>Transfer Accumulator to Index Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TSX</strong></td>
<td>Transfer Stack Pointer to Index X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TXA</strong></td>
<td>Transfer Index X to Accumulator</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TXS</strong></td>
<td>Transfer Index X to Stack Pointer</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TYA</strong></td>
<td>Transfer Index Y to Accumulator</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
THE FOLLOWING NOTATION APPLIES TO THIS SUMMARY:

A Accumulator
X,Y Index Registers
M Memory
C Borrow
P Processor Status Register
S Stack Pointer
✓ Change
– No Change
+ Add
∧ Logical AND
– Subtract
ψ Logical Exclusive OR
↑ Transfer From Stack
→ Transfer To
← Transfer To
V Logical OR
PC Program Counter
PCH Program Counter High
PCL Program Counter low
OPER Operand
# Immediate Addressing Mode

NOTE 1: BIT — TEST BITS
Bit 6 and 7 are transferred 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 B V</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>A - M - C - A C</td>
<td>Immediate</td>
<td>ADC *Oper</td>
<td>69</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>ADC Oper</td>
<td>69</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>ADC Oper</td>
<td>69</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>ADC Oper</td>
<td>69</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>ADC Oper</td>
<td>69</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Indirect(X)</td>
<td>ADC Oper</td>
<td>69</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Indirect(Y)</td>
<td>ADC Oper</td>
<td>69</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>A - M - A</td>
<td>Immediate</td>
<td>AND *Oper</td>
<td>29</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>AND Oper</td>
<td>29</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>AND Oper</td>
<td>29</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>AND Oper</td>
<td>29</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>AND Oper</td>
<td>29</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Indirect(X)</td>
<td>AND Oper</td>
<td>29</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Indirect(Y)</td>
<td>AND Oper</td>
<td>29</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>ASL</td>
<td>Shunt left one bit (Memory or Accumulator)</td>
<td>(See Figure 1)</td>
<td>Accumulator</td>
<td>0A</td>
<td>1</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>ASL Oper</td>
<td>0B</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>ASL Oper</td>
<td>0B</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>ASL Oper</td>
<td>0B</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>ASL Oper</td>
<td>0B</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Indirect(X)</td>
<td>ASL Oper</td>
<td>0B</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Indirect(Y)</td>
<td>ASL Oper</td>
<td>0B</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>BCC</td>
<td>Branch on carry clear</td>
<td>Branch on C = 0</td>
<td>Relative</td>
<td>BCC Oper</td>
<td>90</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>BCS</td>
<td>Branch on carry set</td>
<td>Branch on C = 1</td>
<td>Relative</td>
<td>BCS Oper</td>
<td>90</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>BEQ</td>
<td>Branch on result zero</td>
<td>Branch on Z = 1</td>
<td>Relative</td>
<td>BEQ Oper</td>
<td>90</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>BIT</td>
<td>Test bits in memory with accumulator</td>
<td>A - M: M2 - N, Absolute M2 - Y</td>
<td>BIT* Oper</td>
<td>2A</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>BMI</td>
<td>Branch on result minus</td>
<td>Branch on N = 1</td>
<td>Relative</td>
<td>BMI Oper</td>
<td>30</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>BNE</td>
<td>Branch on result not zero</td>
<td>Branch on Z = 0</td>
<td>Relative</td>
<td>BNE Oper</td>
<td>30</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>BPL</td>
<td>Branch on result plus</td>
<td>Branch on N = 0</td>
<td>Relative</td>
<td>BPL Oper</td>
<td>10</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>BRK</td>
<td>Force Break</td>
<td>Forced Interrupt</td>
<td>BRK*</td>
<td>00</td>
<td>1</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>BVC</td>
<td>Branch on overflow clear</td>
<td>Branch on V = 0</td>
<td>Relative</td>
<td>BVC Oper</td>
<td>50</td>
<td>2</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 B V</th>
</tr>
</thead>
<tbody>
<tr>
<td>BVS</td>
<td>Branch on overflow set</td>
<td>Branch on V = 1</td>
<td>Relative</td>
<td>BVS Oper</td>
<td>70</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>CLC</td>
<td>Clear carry flag</td>
<td>Relative</td>
<td>CLC Oper</td>
<td>18</td>
<td>1</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>CLD</td>
<td>Clear decimal mode</td>
<td>Relative</td>
<td>CLD Oper</td>
<td>08</td>
<td>1</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>CLI</td>
<td>Branch on carry clear</td>
<td>Relative</td>
<td>CLI Oper</td>
<td>58</td>
<td>1</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>CLV</td>
<td>Clear overflow flag</td>
<td>Relative</td>
<td>CLV Oper</td>
<td>88</td>
<td>1</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>CMP</td>
<td>Compare memory and accumulator</td>
<td>A - M</td>
<td>CMP Oper</td>
<td>C9</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>CPX</td>
<td>Compare memory and index X</td>
<td>X - M</td>
<td>CPX Oper</td>
<td>E0</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>CPY</td>
<td>Compare memory and index Y</td>
<td>Y - M</td>
<td>CPY Oper</td>
<td>E4</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>DEC</td>
<td>Decrement memory by one</td>
<td>M - 1</td>
<td>DEC Oper</td>
<td>C6</td>
<td>2</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>DEX</td>
<td>Decrement index X by one</td>
<td>X - 1</td>
<td>DEX Oper</td>
<td>CA</td>
<td>1</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>DEY</td>
<td>Decrement index Y by one</td>
<td>Y - 1</td>
<td>DEY Oper</td>
<td>88</td>
<td>1</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
</tbody>
</table>
### INSTRUCTION CODES

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX OP Code</th>
<th>No. Bys</th>
<th>&quot;P&quot; Status Reg</th>
<th>MZC1D BY</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOR</td>
<td>&quot;Exclusive-Or&quot; memory with accumulator</td>
<td>A VM → A</td>
<td>Immediate</td>
<td>EOR kOp</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page</td>
<td>EOR Op</td>
<td>45 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>EOR Op,X</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute</td>
<td>EOR Op</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,X</td>
<td>EOR Op,X</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,Y</td>
<td>EOR Op,Y</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Indirect,X)</td>
<td>EOR (Op,X)</td>
<td>41 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Indirect,Y)</td>
<td>EOR (Op,Y)</td>
<td>51 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC</td>
<td>Increment memory by one</td>
<td>M → M</td>
<td>Zero Page</td>
<td>INC Op</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>INC Op,X</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute</td>
<td>INC Op</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,X</td>
<td>INC Op,X</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INX</td>
<td>Increment index X by one</td>
<td>X → X</td>
<td>Implied</td>
<td>INX</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INY</td>
<td>Increment index Y by one</td>
<td>Y → Y</td>
<td>Implied</td>
<td>INY</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>Jump to new location</td>
<td>PC-1 → PC</td>
<td>Absolute</td>
<td>JMP Op</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Indirect</td>
<td>JMP (Op)</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSR</td>
<td>Jump to new location saving return address</td>
<td>PC-2 → PC</td>
<td>Absolute</td>
<td>JSR (Op)</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(PC-1) → PCH</td>
<td>JSR (Op)</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA</td>
<td>Load accumulator with memory</td>
<td>M → A</td>
<td>Immediate</td>
<td>LDA kOp</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LDA Op</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>LDA Op,X</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute</td>
<td>LDA Op</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,X</td>
<td>LDA Op,X</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,Y</td>
<td>LDA Op,Y</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Indirect,X)</td>
<td>LDA (Op,X)</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Indirect,Y)</td>
<td>LDA (Op,Y)</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDX</td>
<td>Load index X with memory</td>
<td>M → X</td>
<td>Immediate</td>
<td>LDX kOp</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LDX Op</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page,Y</td>
<td>LDX Op,Y</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute</td>
<td>LDX Op</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,X</td>
<td>LDX Op,X</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,Y</td>
<td>LDX Op,Y</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDY</td>
<td>Load index Y with memory</td>
<td>M → Y</td>
<td>Immediate</td>
<td>LDY kOp</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LDY Op</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>LDY Op,X</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute</td>
<td>LDY Op</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,X</td>
<td>LDY Op,X</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Additional Instructions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX OP Code</th>
<th>No. Bys</th>
<th>&quot;P&quot; Status Reg</th>
<th>MZC1D BY</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSR</td>
<td>Shift right one bit (memory or accumulator)</td>
<td>A VM → A</td>
<td>Immediate</td>
<td>LSR kOp</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LSR Op</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>LSR Op,X</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute</td>
<td>LSR Op</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,X</td>
<td>LSR Op,X</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,Y</td>
<td>LSR Op,Y</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Indirect,X)</td>
<td>LSR (Op,X)</td>
<td>41 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Indirect,Y)</td>
<td>LSR (Op,Y)</td>
<td>51 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>No operation</td>
<td>No Operation</td>
<td>Implied</td>
<td>NOP</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPA</td>
<td>&quot;OP&quot; memory with accumulator</td>
<td>A VM → A</td>
<td>Immediate</td>
<td>OPA kOp</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page</td>
<td>OPA Op</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>OPA Op,X</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute</td>
<td>OPA Op</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,X</td>
<td>OPA Op,X</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,Y</td>
<td>OPA Op,Y</td>
<td>56 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Indirect,X)</td>
<td>OPA (Op,X)</td>
<td>41 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Indirect,Y)</td>
<td>OPA (Op,Y)</td>
<td>51 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHA</td>
<td>Push accumulator on stack</td>
<td>A</td>
<td>Implied</td>
<td>PHA</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHP</td>
<td>Push processor status on stack</td>
<td>P</td>
<td>Implied</td>
<td>PHP</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLA</td>
<td>Pull accumulator from stack</td>
<td>A</td>
<td>Implied</td>
<td>PLA</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLP</td>
<td>Pull processor status from stack</td>
<td>P</td>
<td>Implied</td>
<td>PLP</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROL</td>
<td>Rotate one bit left (memory or accumulator)</td>
<td>(See Figure 2)</td>
<td>Accumulator</td>
<td>ROL kOp</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page</td>
<td>ROL Op</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>ROL Op,X</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute</td>
<td>ROL Op</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,X</td>
<td>ROL Op,X</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,Y</td>
<td>ROL Op,Y</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Indirect,X)</td>
<td>ROL (Op,X)</td>
<td>41 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Indirect,Y)</td>
<td>ROL (Op,Y)</td>
<td>51 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROR</td>
<td>Rotate one bit right (memory or accumulator)</td>
<td>(See Figure 3)</td>
<td>Accumulator</td>
<td>ROR kOp</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page</td>
<td>ROR Op</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>ROR Op,X</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute</td>
<td>ROR Op</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,X</td>
<td>ROR Op,X</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Absolute,Y</td>
<td>ROR Op,Y</td>
<td>46 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Indirect,X)</td>
<td>ROR (Op,X)</td>
<td>41 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Indirect,Y)</td>
<td>ROR (Op,Y)</td>
<td>51 2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

103
### INSTRUCTION CODES

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX BP Code</th>
<th>No. Bytes</th>
<th>P Status Reg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTI</td>
<td>Return from interrupt</td>
<td>P C P C I</td>
<td>Implied</td>
<td>RTI</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>RTS</td>
<td>Return from subroutine</td>
<td>P C P I</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, Op</td>
<td>E9</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>Zero Page, X</td>
<td>SBC, Op</td>
<td>E5</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>SBC, Op</td>
<td>E0</td>
<td>3</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute, Y</td>
<td>SBC, Op</td>
<td>F9</td>
<td>3</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect) X</td>
<td>SBC, (Op), X</td>
<td>E1</td>
<td>2</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect) Y</td>
<td>SBC, (Op), 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>60</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>70</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, Op</td>
<td>85</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page, X</td>
<td>STA, Op, X</td>
<td>95</td>
<td>2</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>STA, Op</td>
<td>80</td>
<td>3</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute, Y</td>
<td>STA, Op, Y</td>
<td>90</td>
<td>3</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect) X</td>
<td>STA, (Op), X</td>
<td>81</td>
<td>2</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect) Y</td>
<td>STA, (Op), Y</td>
<td>91</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, Op</td>
<td>86</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page, Y</td>
<td>STX, Op, Y</td>
<td>96</td>
<td>2</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>STX, Op</td>
<td>8E</td>
<td>3</td>
<td>---</td>
</tr>
<tr>
<td>STY</td>
<td>Store index Y in memory</td>
<td>Y - M</td>
<td>Zero Page</td>
<td>STY, Op</td>
<td>84</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page, X</td>
<td>STY, Op, X</td>
<td>94</td>
<td>2</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>STY, Op</td>
<td>8C</td>
<td>3</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>
### HEX OPERATION CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Instruction</th>
<th>Addressing Mode</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>NOP</td>
<td>00 — BRK</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>ORA</td>
<td>01 — ORA</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>NOP</td>
<td>02 — NOP</td>
<td>(Indirect, Y)</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>NOR</td>
<td>03 — NOR</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>ORA</td>
<td>04 — ORA</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>ZERO PAGE</td>
<td>05 — ORA — ZERO PAGE</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>ASL</td>
<td>06 — ASL — ZERO PAGE</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>NOP</td>
<td>07 — NOP</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>PHP</td>
<td>08 — PHP</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>ORA</td>
<td>09 — ORA</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>0A</td>
<td>ASL</td>
<td>0A — ASL — ZERO PAGE</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>0B</td>
<td>NOP</td>
<td>0B — NOP</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>0C</td>
<td>NOP</td>
<td>0C — NOP</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>0D</td>
<td>ORA</td>
<td>0D — ORA</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>0E</td>
<td>ORA</td>
<td>0E — ORA — Absolute, Y</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>0F</td>
<td>ORA</td>
<td>0F — ORA — Absolute, Y</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>ASL</td>
<td>10 — ASL — Absolute, Y</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>ORA</td>
<td>11 — ORA — (Indirect), Y</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>NOP</td>
<td>12 — NOP</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>NOP</td>
<td>13 — NOP</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>NOP</td>
<td>14 — NOP</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>ORA</td>
<td>15 — ORA — Zero Page, X</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>ASL</td>
<td>16 — ASL — Zero Page, X</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>NOA</td>
<td>17 — NOR</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>CLC</td>
<td>18 — CLC</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>ORA</td>
<td>19 — ORA — Absolute, Y</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>1A</td>
<td>ORA</td>
<td>1A — ORA — Absolute, X</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>1B</td>
<td>ORA</td>
<td>1B — ORA — Absolute, X</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>1C</td>
<td>ORA</td>
<td>1C — ORA — Absolute, X</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>1D</td>
<td>ASL</td>
<td>1D — ASL — Absolute, X</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>1E</td>
<td>ASL</td>
<td>1E — ASL — Absolute, X</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>1F</td>
<td>NOP</td>
<td>1F — NOP</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>JSR</td>
<td>20 — JSR</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>AND</td>
<td>21 — AND — (Indirect), X</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>ORA</td>
<td>22 — ORA</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>NOP</td>
<td>23 — NOP</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>BIT</td>
<td>24 — BIT — Zero Page</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>AND</td>
<td>25 — AND — Zero Page</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>ROL</td>
<td>26 — ROL — Zero Page</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>NOP</td>
<td>27 — NOP</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>PLP</td>
<td>28 — PLP</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>AND</td>
<td>29 — AND — Immediate</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>2A</td>
<td>ROL</td>
<td>2A — ROL — Accumulator</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>2B</td>
<td>NOP</td>
<td>2B — NOP</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>2C</td>
<td>BIT</td>
<td>2C — BIT — Absolute</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>2D</td>
<td>AND</td>
<td>2D — AND — Absolute</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
<tr>
<td>2E</td>
<td>ROL</td>
<td>2E — ROL — Absolute</td>
<td>(Indirect, X)</td>
<td></td>
</tr>
</tbody>
</table>

Note: The table continues with more codes and descriptions, covering a range of operations and addressing modes.
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
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:

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

- 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 10mA.

  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.

j. 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 mother board, 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 "@" (shift-P) to clear screen. You may now try "Monitor" commands if you wish. See details in "Monitor" 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 indefinitely 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.
NOTES ON INTERFACING WITH THE HOME TV

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 an 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 50 dB 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 500 μV 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 $300 hex of memory. This program resides in memory from $370 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 $370 to $3E9. For example, type in "370: A9 82" and then depress and release the "RETURN" key. Then repeat this procedure for the data at $372 and on until you complete entering the program.

Executing this Program

1. From BASIC a CALL 880 ($370) will start the execution of this program. It will use the teletype or suitable 80 column printer as the primary output device.
2. PRØØ will inactivate the printer transferring control back to the Video monitor as the primary output device.

3. In Monitor mode $37ØG 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 asterick "*" 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 88Ø
   15 PRINT "ABCD...XYZØ1123456789"
   20 PRØØ
   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 1Ø 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 2Ø 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.
FIGURE 2  RS-232

+12 (JUMPERED TO +12 SUPPLY)

2N3906

3.3K

PIN 15

J-14

OUT(-)

3.3K

2.2K

PIN 8

J-14

-12 (JUMPERED TO -12 SUPPLY)

FIGURE 2  ASR-33

(a) 

(b) 

FIGURE 2  RS-232

OUTPUT TO TELETYPES

3.3K

150Ω

RESISTORS ARE 1/4 WATT CARBON

2N3906 (OR EQUIV.)

+5V

PIN 15

J-14
FIGURE 3a

TELETYPe DRIVER ROUTINES

3:42 P.M., 11/18/1977

1 TITLE TELETYPe DRIVER ROUTINES'

2 ****************************

3 *   TTYDRIVER: *

4 *   TELETYPe OUTPUT *

5 *   ROUTINE FOR 72 *

6 *   COLUMN PRINT WITH*

7 *   BASIC LIST *

8 *   *

9 *   *

10 *   COPYRIGHT 1977 BY: *

11 *   APPLE COMPUTER INC.*

12 *   11/18/77 *

13 *   *

14 *   R. WIGGINTON *

15 *   S. WOZNIK *

16 *   *

17 ****************************

18 WNDWDTH EQU $21 ;FOR APPLE-II

19 CH EQU $24 ;CURSOR HORIZ.

20 CSWL EQU $36 ;CHAR. OUT SWITCH

21 YSAVE EQU $778

22 COLCNT EQU $7F8 ;COLUMN COUNT LOC.

23 MARK EQU $CO58

24 SPACE EQU $CO59

25 WAIT EQU $FCA8

26 ORG $370

***WARNING: OPERAND OVERFLOW IN LINE 27

0370: A9 82     27 TTINIT: LDA #$TOUT

0372: 85 36     28 STA CSWL ;POINT TO TTY ROUTINES

0374: A9 03     29 LDA #$TOUT/256 ;HIGH BYTE

0376: 85 37     30 STA CSWL+1

0378: A9 48     31 LDA #72 ;SET WINDOW WIDTH

037A: 85 21     32 STA WNDWDTH ;TO NUMBER COLUMNS ONT

037C: A5 24     33 LDA CH

037E: 8D F8     34 STA COLCNT ;WHERE WE ARE NOW.

0381: 60 35     35 RTS

0382: 48 36     36 TTOUT: PHA ;SAVE TWICE

0383: 48 37     37 PHA ;ON STACK.

0384: AD F8     38 TTOUT2: LDA COLCNT ;CHECK FOR A TAB.

0387: C5 24     39 CMP CH

0389: 68 40     40 PLA ;RESTORE OUTPUT CHAR.

038A: BO 03     41 BCS TESTCTRL ;IF C SET, NO TAB

038C: 48 42     42 PHA

038D: A9 A0     43 LDA #$A0 ;PRINT A SPACE.

038F: 2C CO     44 TESTCTRL BITRTS1 ;TRICK TO DETERMINE

0392: FO 03     45 BEQ PRNTIT ;IF CONTROL CHAR.

0394: EE F8     46 INC COLCNT ;IF NOT, ADD ONE TO CM

0397: 20 C1     47 PRNTIT: JSR DOCHAR ;PRINT THE CHAR ON TTY

039A: 68 48     48 PLA ;RESTORE CHAR

0393: 48 49     49 PHA TTOUT2 ;AND PUT BACK ON STAC

039C: 90 E6     50 BCC #$OD ;DO MORE SPACES FOR TA

039E: 49 OD     51 FOR A ;CHECK FOR CAR RET.

03A0: OA 52     52 ASL FINISH ;ELIM PARITY

03A1: DO OD     53 BNE ;IF NOT CR, DONE.
TELETYPewriter DRIVER ROUTINES

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 FINISH:
0333: F0 08 60 LDA COLCNT ;CHECK IF IN MARGIN
0335: E5 21 61 S3C 7V7D7TH
0337: E9 F7 62 SSC #SF7
0339: 90 04 63 BCC RETURN
0393: 69 1F 64 ADC #11F ;ADJUST CH
039D: 85 24 65 STA CH
033D: 68 66 SETCH: PLA
03C0: 60 67 RETURN: RTS ;RETURN TO CALLER
03C1: 68 66 RTS1: RTS
03C4: 8C 78 07 69 * HERE DOCHAR:
03C5: 08 70 PHP YSAVE
03C7: A0 08 71 A character routine:
03C8: 18 72 CLC #SOS
03C9: 48 73 11 BITS (1 START, 1 2
03C3: 80 05 74 SEND A SPACE
03CE: AD 59 C0 75 110 BAUD
0300: 90 03 76 LOOP 11 3ITS.
0303: AD 58 C0 77
0305: A9 D7 78 MARKOUT:
0306: 48 79 LDA TTOUT4 ;SEND A MARK
0308: A9 20 80 TTOUT4: LDA
0309: 4A 81 DLY1: LSR #$20
030D: 90 FD 82 DLY2: BCC A
03DC: 68 83 PLA DLY2
030E: 6A 84 SBC
03BE: 88 85 3NE #101
03B0: 88 85 PLA DLY1
03B1: D0 E3 86 ROR ;110 BAUD
03B2: AC 78 07 87 DEY A ;NEXT BIT (STOP BITS ?
03B3: 28 88 BNE LOOP 11 3ITS.
03B5: 60 89 LDY TTOUT3
03B8: 90 PLP YSAVE ;RESTORE Y-REG.
03B9: 91 RTS ;RESTORE STATUS

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

FIGURE 3b
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
Figure 1B Connector Location Detail

Front of PC Board

Right Side of PC Board

Back Edge of PC Board

Peripherals

Top View

APPLE II PC BOARD

CONNECTOR LOCATIONS

KEYBOARD CONNECTOR

POWER CONNECTOR

BACK EDGE OF PC BOARD
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. $V_{IN}=1V_{pp}$ (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. $V_{OUT}=25mV$ into 100 Ohms, $Z_{OUT}=100$ Ohms. Located at K13 as illustrated in 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.

Figure 2

GAME I/O CONNECTOR

(TOP VIEW)

(Front Edge of PC Board)

+5V 1 16 N.C.
SWO 2 15 ANO
SW1 3 14 AN1
SW2 4 13 AN2
CO4O STB 5 12 AN3
PDLO 6 11 PDL3
PDL2 7 10 PDL1
GND 8 9 N.C.

LOCATION J14
**SIGNAL DESCRIPTIONS FOR GAME I/O**

**ANO-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.

**C040 STB:**
A utility strobe output. Will go low during Øo 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 Ø150K 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>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+5V</td>
</tr>
<tr>
<td>2</td>
<td>STROBE</td>
</tr>
<tr>
<td>3</td>
<td>RESET</td>
</tr>
<tr>
<td>4</td>
<td>N.C.</td>
</tr>
<tr>
<td>5</td>
<td>B6</td>
</tr>
<tr>
<td>6</td>
<td>B5</td>
</tr>
<tr>
<td>7</td>
<td>B7</td>
</tr>
<tr>
<td>8</td>
<td>GND</td>
</tr>
<tr>
<td>9</td>
<td>N.C.</td>
</tr>
<tr>
<td>10</td>
<td>B3</td>
</tr>
<tr>
<td>11</td>
<td>B4</td>
</tr>
<tr>
<td>12</td>
<td>B1</td>
</tr>
<tr>
<td>13</td>
<td>B2</td>
</tr>
<tr>
<td>14</td>
<td>N.C.</td>
</tr>
<tr>
<td>15</td>
<td>-12V</td>
</tr>
<tr>
<td>16</td>
<td>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 #2HW25C0-11l (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

A0-A15: 16 bit system address bus. Addresses are set up by the 6502 within 300nS after the beginning of Ø1. 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 Ø2 (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 Ø2. During a read cycle the 6502 expects data to be ready no less than 100nS before the end of Ø2. These lines will drive up to a total of 8 standard 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 Ø2, of a read or write to any address C800-CFFF. 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.

Ø: 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 #9-35Ø28-1,6 pin connector. See location and pin out in Figures 1 and 5.

PIN DESCRIPTION

GND: (2 pins) system circuit ground. Ø 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)

PINOUT
TOP VIEW
(Back Edge of PC Board)

LOCATIONS J2 TO J12

Figure 5  POWER CONNECTOR

PINOUT
TOP VIEW
(Toward Right side of PC Board)

LOCATION K1
SPEAKER CONNECTOR

This is a MOLEX KK 100 series connector with two .25" square pins on .10" 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.

Figure 6

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 0 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. Ø Volt line from power supply.

VIDEO: NTSC compatible positive composite VIDEO. DC coupled emitter follower output (not short circuit protected). SYNC TIP is Ø 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

- GND
- VIDEO
- -5V
- +12V

Right 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

TOP VIEW

PINOUT

(0000-0FFF) 4K "0" BLOCK1
(1000-1FFF) 4K "1" BLOCK2
(2000-2FFF) 4K "2" BLOCK3
(3000-3FFF) 4K "3" BLOCK4
(4000-4FFF) 4K "4" BLOCK5
(5000-5FFF) 4K "5" BLOCK6
(6000-EFFF) 4K "6" BLOCK7

14 RAM ROW C
13 RAM ROW D
12 RAM ROW E
11 N.C.
10 16K "0" BLOCK (0000-3FFF)
 9 16K "4" BLOCK (4000-7FFF)
 8 16K "8" BLOCK (8000-BFFF)

LOCATIONS D1, E1, F1

133
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, 20K 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, Thcated 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-C1Ø on the board), and you want them to occupy the first 16K of addresses starting at 0000, jumper pin 14 to pin 1Ø 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</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000-001F</td>
<td>UTILITY</td>
<td>register area for &quot;sweet 16&quot; 16 bit firmware processor.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MONITOR</td>
<td>holds a 16 bit number that is randomized with each key entry.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BASIC</td>
<td>integer multiply and divide work space.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UTILITY</td>
<td>floating point work space.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6502</td>
<td>subroutine return stack.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>character input buffer.</td>
<td></td>
</tr>
<tr>
<td>PAGE ONE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100-01FF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAGE TWO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0200-02FF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAGE THREE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03FE-03FF</td>
<td></td>
<td></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 Ø to BFF

- Addresses dedicated to hardware functions
- ROM socket DO: spare
- ROM socket DS: spare
- ROM socket EO: BASIC
- ROM socket ES: BASIC
- ROM socket FO: BASIC
- ROM socket FS: monitor

BASIC initializes LOREM 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;C040 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.</td>
</tr>
<tr>
<td>C061/9</td>
<td>&quot;SW1&quot;</td>
<td>State of Switch 1 Game I/O connector appears in bit 7.</td>
</tr>
<tr>
<td>C062/A</td>
<td>&quot;SW2&quot;</td>
<td>State of Switch 2 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 Switch 3 input on Game I/O connector appears in bit 7.</td>
</tr>
<tr>
<td>C064/C</td>
<td>Paddle 0 timer output</td>
<td>State of timer output for Paddle 0 appears in bit 7.</td>
</tr>
<tr>
<td>C065/D</td>
<td>&quot;1&quot;</td>
<td>State of timer output for Paddle 1 appears in bit 7.</td>
</tr>
<tr>
<td>C066/E</td>
<td>&quot;2&quot;</td>
<td>State of timer output for Paddle 2 appears in bit 7.</td>
</tr>
<tr>
<td>C067/F</td>
<td>&quot;3&quot;</td>
<td>State of timer output for Paddle 3 appears in bit 7.</td>
</tr>
<tr>
<td>C07X</td>
<td>&quot;PDL STB&quot;</td>
<td>Triggers paddle timers during $\varphi_2$.</td>
</tr>
<tr>
<td>C08X</td>
<td>DEVICE SELECT 0</td>
<td>Pin 41 on the selected Peripheral Connector goes low during $\varphi_2$.</td>
</tr>
<tr>
<td>C09X</td>
<td>&quot;1&quot;</td>
<td></td>
</tr>
<tr>
<td>C0AX</td>
<td>&quot;2&quot;</td>
<td></td>
</tr>
<tr>
<td>C0BX</td>
<td>&quot;3&quot;</td>
<td></td>
</tr>
<tr>
<td>C0CX</td>
<td>&quot;4&quot;</td>
<td></td>
</tr>
<tr>
<td>C0DX</td>
<td>&quot;5&quot;</td>
<td></td>
</tr>
<tr>
<td>C0EX</td>
<td>&quot;6&quot;</td>
<td></td>
</tr>
<tr>
<td>C0FX</td>
<td>&quot;7&quot;</td>
<td></td>
</tr>
<tr>
<td>C10X</td>
<td>&quot;8&quot;</td>
<td></td>
</tr>
<tr>
<td>C11X</td>
<td>&quot;9&quot;</td>
<td>Expansion connectors.</td>
</tr>
<tr>
<td>C12X</td>
<td>&quot;A&quot;</td>
<td></td>
</tr>
</tbody>
</table>

138
<table>
<thead>
<tr>
<th>HEX ADDRESS</th>
<th>ASSIGNED FUNCTION</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C13X</td>
<td>DEVICE SELECT B</td>
<td></td>
</tr>
<tr>
<td>C14X</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C15X</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C16X</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C17X</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C1XX</td>
<td>I/O SELECT 1</td>
<td>Pin 1 on the selected Peripheral Connector goes low during $\phi_2$.</td>
</tr>
<tr>
<td>C2XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C3XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C4XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C5XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C6XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C7XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C8XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C9XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>CAXX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>CBXX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>CCXX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>CDXX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>CEXX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>CFXX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>D000-D7FF</td>
<td>ROM socket D0</td>
<td>Spare.</td>
</tr>
<tr>
<td>D800-DFFF</td>
<td>&quot;</td>
<td>Spare.</td>
</tr>
<tr>
<td>E000-E7FF</td>
<td>&quot;</td>
<td>BASIC.</td>
</tr>
<tr>
<td>E800-EFFF</td>
<td>&quot;</td>
<td>BASIC.</td>
</tr>
<tr>
<td>F000-F7FF</td>
<td>&quot;</td>
<td>1K of BASIC, 1K of utility.</td>
</tr>
<tr>
<td>F800-FFFFF</td>
<td>&quot;</td>
<td>Monitor.</td>
</tr>
</tbody>
</table>
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, 3.580 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.

Ø3: 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

[Timings diagram with labels for 14M, 7M, COLOR REF, Ø0, Ø1, Ø2, Ø3, timing relationships]
FIGURE S-1  APPLE II SYSTEM DIAGRAM
FIGURE S-3 REFERENCE OSCILLATOR AND SYSTEM TIMING
FIGURE S-4  SYNC COUNTER
FIGURE S-7 RAM ADDRESS MUX
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