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

Current Donators to this project are:
Dan Chisarick, David Schmidt
Apple II Reference Manual
January 1978
# 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
- 13. Additional BASIC Program Examples  55
  - 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

**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  122
- 5. Interfacing the APPLE - Signals, Loading, Pin Connections  122
- 6. Memory - Options, Expansion, Map, Address  133
- 7. System Timing  140
- 8. Schematics  141
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 (minature phone jack type)

3. If you purchased a 16K or larger system, your accessory box should also contain:
   a. 16K Star trek 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:

A2SØØMMX

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

A2SØØ8X

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 undistorted 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. \texttt{>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 anout 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 straightforward: 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 color graphics 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
PROGRAM LISTING

```
5 GOTO 15
10 Q=(PDL(0)-20)/6: IF Q<0 THEN
 Q=0: IF Q>=34 THEN Q=34: COLOR=D:
 VLIN Q,Q+5 AT 0: COLOR=A:
 IF P>Q THEN 175: IF Q THEN
 VLIN 0,Q-1 AT 0:P=Q: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$: INPUT "Y/N? ",B$: GR: CALL -936:
 IF B$(1,1)#"N" THEN 40
 FOR I=0 TO 39: COLOR=I/2*(I(32)
 30 NEXT I: POKE 34,20: PRINT :
 PRINT: FOR I=0 TO 15: VTAB 21+I MOD 2:
 TAB I+I+1: PRINT I;: NEXT I: POKE
 34,22: YTAB 24: PRINT
 "BACKGROUND"; 35 GOSUB 95: AHE: PRINT "EVEN BRICK"
 ;GOSUB 95: BHE: PRINT "ODD BRIC
 K"; GOSUB 95: CHE: PRINT "PADDLE"
 ;GOSUB 95: DHE: PRINT "BALL";
 GOSUB 95
40 POKE 34,28: COLOR=A: FOR I=0
 TO 39: VLIN B,39 AT I: NEXT I:
 FOR I=20 TO 34 STEP 2: TAB
 I+1: PRINT I/2-9;: COLOR=8:
 VLIN I,39 AT I: NEXT I: TAB
 6: PRINT "SCORE =0": PRINT
 55 IF L=1 THEN PRINT 'LAST BALL, '
 ;A$: PRINT: FOR I=1 TO 100
 : GOSUB 16: NEXT I:K=1:H=0
 60 J=Y+H: IF J=0 AND I<120 THEN
 65 I=I-1: I=I: FOR I=1 TO 6:K=
 PEEK(-16336): NEXT I
 65 I=I+H: IF I=0 THEN 100:
 GOSUB 16: NEXT I
 70 Z=PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336): GOTO 60
 75 FOR I=1 TO 6:H=PEEK(-16336)
 <H>: NEXT I:K=H
 80 Y=V
 85 POKE K,Y/3: COLOR=E: POKE I,
 K=I+1=I: GOTO 60
 90 PRINT "INVALID, REENTER";
 95 INPUT " COLOR (0, TO 15)",E:
 IF E=0 OR E>15 THEN 98: RETURN
100 IF M THEN W=ABS(Y): VLIN
 W/2+X,W/2+Y AT I: S=W=S+I/2-
 9: YTAB 21: TAB 13: PRINT S
 105 O=PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336)-PEEK(-16336)
 110 IF S<720 THEN 80
 115 PRINT "CONGRATULATIONS, ";A$: 
 "YOU WIN!": GOTO 165
 120 PRINT "YOUR SCORE OF ";S; IS 
 ;; GOTO 125+(S/100)*5
 125 PRINT "TERRIBLE!": GOTO 165
 130 PRINT "LOUSY.": GOTO 165
 135 PRINT "POOR.": GOTO 165
 140 PRINT "GOOD.": GOTO 165
 145 PRINT "VERY GOOD.": GOTO 165
 155 PRINT "EXCELLENT.": GOTO 165
 160 PRINT "NEARLY PERFECT.";
 165 PRINT "ANOTHER GAME ";A$; "(Y/N)
 ;: INPUT A$: IF A$(1,1)="Y"
 THEN 25: TEXT: CALL -936:
 VTab 10: TAB 10: PRINT 
 "GAME OVER": END
 170 Z=PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336): GOTO 60
 175 FOR I=1 TO 80:Q=PEEK(-16336)
 <Q>: NEXT I:Q=Q
 180 FOR I=1 TO 90:O=PEEK(-16336)
 >O: NEXT I:O=O
```

---

**Breakout Game**

**Program Listing**

```
5 GOTO 15
10 Q=(PDL(0)-20)/6: IF Q<0 THEN
 Q=0: IF Q>=34 THEN Q=34: COLOR=D:
 VLIN Q,Q+5 AT 0: COLOR=A:
 IF P>Q THEN 175: IF Q THEN
 VLIN 0,Q-1 AT 0:P=Q: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$: INPUT "Y/N? ",B$: GR: CALL -936:
 IF B$(1,1)#"N" THEN 40
 FOR I=0 TO 39: COLOR=I/2*(I(32)
 30 NEXT I: POKE 34,20: PRINT :
 PRINT: FOR I=0 TO 15: VTAB 21+I MOD 2:
 TAB I+I+1: PRINT I;: NEXT I: POKE
 34,22: YTAB 24: PRINT
 "BACKGROUND"; 35 GOSUB 95: AHE: PRINT "EVEN BRICK"
 ;GOSUB 95: BHE: PRINT "ODD BRIC
 K"; GOSUB 95: CHE: PRINT "PADDLE"
 ;GOSUB 95: DHE: PRINT "BALL";
 GOSUB 95
40 POKE 34,28: COLOR=A: FOR I=0
 TO 39: VLIN B,39 AT I: NEXT I:
 FOR I=20 TO 34 STEP 2: TAB
 I+1: PRINT I/2-9;: COLOR=8:
 VLIN I,39 AT I: NEXT I: TAB
 6: PRINT "SCORE =0": PRINT
 55 IF L=1 THEN PRINT 'LAST BALL, '
 ;A$: PRINT: FOR I=1 TO 100
 : GOSUB 16: NEXT I:K=1:H=0
 60 J=Y+H: IF J=0 AND I<120 THEN
 65 I=I-1: I=I: FOR I=1 TO 6:K=
 PEEK(-16336): NEXT I
 65 I=I+H: IF I=0 THEN 100:
 GOSUB 16: NEXT I
 70 Z=PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336): GOTO 60
 75 FOR I=1 TO 6:H=PEEK(-16336)
 <H>: NEXT I:K=H
 80 Y=V
 85 POKE K,Y/3: COLOR=E: POKE I,
 K=I+1=I: GOTO 60
 90 PRINT "INVALID, REENTER";
 95 INPUT " COLOR (0, TO 15)",E:
 IF E=0 OR E>15 THEN 98: RETURN
100 IF M THEN W=ABS(Y): VLIN
 W/2+X,W/2+Y AT I: S=W=S+I/2-
 9: YTAB 21: TAB 13: PRINT S
 105 O=PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336)-PEEK(-16336)
 110 IF S<720 THEN 80
 115 PRINT "CONGRATULATIONS, ";A$: 
 "YOU WIN!": GOTO 165
 120 PRINT "YOUR SCORE OF ";S; IS 
 ;; GOTO 125+(S/100)*5
 125 PRINT "TERRIBLE!": GOTO 165
 130 PRINT "LOUSY.": GOTO 165
 135 PRINT "POOR.": GOTO 165
 140 PRINT "GOOD.": GOTO 165
 145 PRINT "VERY GOOD.": GOTO 165
 155 PRINT "EXCELLENT.": GOTO 165
 160 PRINT "NEARLY PERFECT.";
 165 PRINT "ANOTHER GAME ";A$; "(Y/N)
 ;: INPUT A$: IF A$(1,1)="Y"
 THEN 25: TEXT: CALL -936:
 VTab 10: TAB 10: PRINT 
 "GAME OVER": END
 170 Z=PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336)-PEEK(-16336)
 +PEEK(-16336): GOTO 60
 175 FOR I=1 TO 80:Q=PEEK(-16336)
 >Q: NEXT I:Q=Q
 180 FOR I=1 TO 90:O=PEEK(-16336)
 >O: NEXT I:O=O
```
COLOR DEMO PROGRAM

LISTING

10 DIM C(4); POKE 2,173: POKE
3,48; POKE 4,192; POKE 5,165
; POKE 6,8; POKE 7,32; POKE
8,168; POKE 9,252; POKE 10,
165; POKE 11,1; POKE 12,268
20 POKE 13,4; POKE 14,198; POKE
15,24; POKE 16,240; POKE 17
,5; POKE 18,198; POKE 19,1;
POKE 20,78; POKE 21,2; POKE
22,0; POKE 23,36
30 TEXT; CALL -936; VTB4; 4: TAB
3: PRINT "4K COLOR DEGOS": 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
#",1: GR: IF I=0 AND I<11
THEN GOTO 100+1: GOTO 30
70 INPUT "WHICH DEMO WOULD YOU LIKE
",1: GR: IF I AND I<20 THEN
GOTO 100+1: GOTO 30
100 I=I+1 MOD 79;j=I*(1339)*79
-1-J): GOSUB 2000: GOSUB 10000
; GOTO 100
200 I=I+1 MOD 39;J=I: GOSUB 2000
; J=39-I: GOSUB 2000: GOSUB
10000: GOTO 200
300 J=J+1:J=1 MOD 22+1: FOR I=1
TO 1295: COLOR=I MOD 7: PLOT
(2*J) MOD 37,(3*I) MOD 35: NEXT
I: GOSUB 10000: GOTO 300
400 FOR I=1 TO 4:C(I)= RND(16)
; NEXT I
500 FOR I=0 TO 1 STEP -1:C(I+1)
; =C(I): NEXT I; C(I)= RND(16)
; FOR I=1 TO 5: FOR J=1 TO 4
600 COLOR=C(J);L=J*5+14+I*33-
L: HLIN K,L AT K; VLIN K,L AT
L: HLIN K,L AT L; VLIN K,L AT
K: NEXT J;I: GOSUB 10000: GOTO
410
700 I=I+1: RHD(36)+J=I*15+I*26+
700+32767/J< PDL (0)/10):
POKE 0,1: POKE 1,1 MOD 256
; POKE 24,(K+255)+1: CALL 2
; GOSUB 10000: GOTO 700
800 X=I+I:POKE 78=X;POKE 79=4;Y=0
; FOR I=1: COLOR=6; HLIN 0,39 AT
4: COLOR=9: GOSUB 880; COLOR=12:
VLI N 5,M AT X
900 N=2*P/P-A/W: COLOR=0: GOSUB
880; VLI N 5,39 AT X:X=X+1: IF
X<39 THEN 826;X=3: VLI N 5,39
AT X: VLI N 5,39 AT 2
910 P=RHD(36)+J=I*100: COLOR=12: GOSUB
880; COLOR=9: VLI N 5,M AT X:
X: COLOR=15: PLOT X-2,M: FOR
I=0 TO J: NEXT I: GOSUB 10000
; GOTO 810
920 M=L-Y:LI=H-1:L=2+H+1: VLI N L1,
L2 AT X-1: VLI N L1 L2 AT X:
VLI N L1 L2 AT X+1: RETURN
930 I=I+1 MOD 15: FOR Y=0 TO 33
; FOR X=0 TO 39: COLOR=I+1 ABS
(29-X)-Z: ABS (29-Y)-Z:25
; PLOT X,Y: NEXT X,Y: GOSUB
10000: GOTO 900
1000 CALL -936
1100 J=I+1 MOD 32: IF J=32: YL
H,39 AT 3: VTB4 (J+2) MOD
2: TAB 3+J: IF J MOD 2 THEN
PRINT J;GOSUB 10000: GOTO
1100
1200 COLOR= RND(16); HLIN 0,39 AT
J; COLOR= RND(16); VLI N 0,
39 AT J: RETURN
1300 IF PEEX (-16384)*128 THEN RETURN
; POKE -16368,0: POP : GOTO
30
THIS IS A SHORT DESCRIPTION OF HOW TO PLAY STARTREK ON THE
APPLE COMPUTER.

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

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

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

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

TO PLAY:

1. THE COMMANDS CAN BE OBTAINED BY TYPING A '0' (ZERO) AND RETURN.
   THEY ARE:
   1. PROPULSION
   2. REGENERATE
   3. LONG RANGE SENSORS
   4. PHASERS
   5. PHOTON TORPEDOES
   6. GALAXY RECORD
   7. COMPUTER
   8. PROBE
   9. SHIELD ENERGY
   10. DAMAGE REPORT
   11. LOAD PHOTON TORPEDOES

2. THE COMMANDS ARE INVOKED BY TYPING THE NUMBER REFERING TO THEM
   FOLLOWED BY A 'RETURN'.
   A. IF RESPONSE IS 1 THE COMPUTER WILL ASK WARP OR ION AND
      EXPECTS 'W' IF ONE WANTS TO TRAVEL IN THE GALAXY
      BETWEEN QUADRANTS AND AN 'I' IF ONE WANTS ONLY
      INTERNAL QUADRANT TRAVEL.
      DURATION OF WARP FACTOR IS THE NUMBER OF SPACES OR
      QUADRANTS THE ENTERPRISE WILL MOVE.
      COURSE IS COMPASS READING IN DEGREES FOR THE DESI-
      RED DESTINATION.
   B. A 2 REGENERATES THE ENERGY AT THE EXPENSE OF TIME.
   C. A 3 GIVES THE CONTENTS OF THE IMMEDIATE. ADJACENT QUADRANTS.
      THE GALAXY IS WRAP-AROUND IN ALL DIRECTIONS.
   D. A 4 FIRES PHASERS AT THE EXPENSE OF AVAILABLE ENERGY.
   E. 5 INITIATES A SET OF QUESTIONS FOR TORPEDO FIRING.
      THEY CAN BE FIRED AUTOMATICALLY IF THEY HAVE
      BEEN LOCKED ON TARGET WHILE IN THE COMPUTER
      MODE, OR MAY BE FIRED MANUALLY IF THE TRAJECTORY ANGLE
      IS KNOWN.
   F. 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 RESPOND 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 FIRST FIVE ONE WILL HAVE TO GIVE COORDINATES.
      COORDINATES ARE GIVEN IN MATHEMATICAL NOTATION WITH
      THE EXCEPTION THAT THE 'Y' VALUE IS GIVEN FIRST.
      AN EXAMPLE WOULD BE 'Y,X'
      COURSE OR TREJECTORY:

     0
      ---------------
     270
      ---------------
    180

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

PROCEDURE

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

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

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

4. Type in "C00.FFFR" on the Apple II keyboard. This is the address range of the high resolution machine language 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:
destroyed current variable values.

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

NEW
Clears (Scratches) current BASIC program.

NO DSP var
Clears DSP mode for variable var.

NO TRACE
Clears TRACE mode.

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

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

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

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

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

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

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

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

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

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

FUNCTION NAME

<table>
<thead>
<tr>
<th>FUNCTION NAME</th>
<th>300 PRINT ABS(X)</th>
<th>Gives absolute value of the expression expr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS (expr)</td>
<td>310 PRINT ASC(&quot;BACK&quot;)</td>
<td>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.</td>
</tr>
<tr>
<td>ASC (str$)</td>
<td>320 PRINT ASC(B$)</td>
<td>Gives current length of designated string variable str$; i.e., number of characters.</td>
</tr>
<tr>
<td>LEN (str$)</td>
<td>330 PRINT ASC(B$(4,4))</td>
<td>Gives number between Ø and 255 corresponding to paddle position on game paddle number designated by expression expr and must be legal paddle (Ø,1,2,or 3) or else 255 is returned.</td>
</tr>
<tr>
<td>PDL (expr)</td>
<td>340 PRINT ASC(B$(Y))</td>
<td>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</td>
</tr>
<tr>
<td>PEEK (expr)</td>
<td>350 PRINT PDL(X)</td>
<td>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).</td>
</tr>
<tr>
<td>RND (expr)</td>
<td>360 PRINT PEEK(X)</td>
<td>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 colorgraphics display mode, or Ø to 47 if in all color mode set by GR command or Ø to 39 if in all color mode set by POKE -163Ø4,Ø; POKE -163Ø2,Ø'.</td>
</tr>
<tr>
<td>SCRBN(expr1, expr2)</td>
<td>380 PRINT SCRN (X1,Y1)</td>
<td>Gives the sign (not sine) of expression expr i.e., -1 if expression expr is negative, zero and +1 if expr is positive.</td>
</tr>
</tbody>
</table>
BASIC STATEMENTS

Each BASIC statement must have a line number between 0 and 32767. Variable names must start with an alpha character and may be any number of 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 (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 \( expr \) 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.

DIM str$(expr2)
60 DIM B$(30)

DIM vari2(expr3)
70 DIM C (2)

LEGAL:
80 DIM A(30)

LEGAL:
85 DIM C(1000)

DSP vari

Legal:
90 DSP AX: DSP L
Illegal:
100 DSP AX,B
102 DSP AB$
104 DSP A(5)

Legal:
105 A=A(5): DSP A

Sets debug mode that DSP variable vari each time it changes and the line number where the change occurred.
<table>
<thead>
<tr>
<th>NAME</th>
<th>EXAMPLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>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 and 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>STEP</td>
<td>120 FOR X=Y1 TO Y3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130 FOR 1=39 TO 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>150 GOSUB 100 *J2</td>
<td></td>
</tr>
<tr>
<td>GOSUB</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>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 and expr2 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 is true (non-zero) then execute statement; if false do not execute statement. If statement is an expression, then a GOTO expr type of statement is assumed to be implied. The "ELSE" in example 26Ø is illegal but may be implemented as shown in example 27Ø.

Enters 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.

Transfers source of data for subsequent INPUT statements to peripheral I/O slot (1-7) as specified as by expression expr. Slot Ø is not addressable from BASIC. IN#Ø (Example 33Ø) is used to return data source from peripherial I/O to keyboard connector.

Assignment operator. "LET" is optional

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

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

Turns-off DSP debug mode for variable

Turns-off TRACE debug mode
PLOT expr1, expr2

400 PLOT 15, 25
400 PLT XV,YV

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

POKE expr1, expr2

420 POKE 20, 40
430 POKE 7*256, XMOD255

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

POOP

440 POP

"POPS" nested GOSUB return stack address by one.

PRINT var1, var, str$

450 PRINT L1
460 PRINT Li, X2
470 PRINT "AMT=":DX
480 PRINT A$;B$;
490 PRINT
492 PRINT "HELLO"
494 PRINT 2+3

Outputs data specified by variable var or string variable str$ starting at current cursor location. If there is not trailing "," or ";" (Ex 450) a carriage return will be generated. Commas (Ex. 460) outputs data in 5 left justified columns. Semi-colon (Ex. 470) inhibits print of any spaces. Text imbedded in " " will be printed and may appear multiple times.

PR# expr

500 PR# 7

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

REM

510 REM REMARK

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

RETURN

520 RETURN
530 IF X= 5 THEN RETURN

Causes branch to statement following last GOSUB; i.e., RETURN ends a subroutine. Do not confuse "RETURN" statement with Return key on keyboard.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TAB</strong> (\text{expr})</td>
<td>Moves cursor to absolute horizontal position specified by expression (\text{expr}) in the range of 1 to 40. Position is left to right.</td>
</tr>
<tr>
<td><strong>TEXT</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>TRACE</strong></td>
<td>Sets debug mode that displays each line number as it is executed.</td>
</tr>
<tr>
<td><strong>VLIN</strong> (\text{expr}_1, \text{expr}_2) AT (\text{expr}_3)</td>
<td>Similar to HLIN except draws vertical line starting at (\text{expr}_1) and ending at (\text{expr}_2) at horizontal position (\text{expr}_3).</td>
</tr>
<tr>
<td><strong>VTAB</strong> (\text{expr})</td>
<td>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.</td>
</tr>
</tbody>
</table>
SPECIAL CONTROL AND EDITING CHARACTERS

"Control" characters are indicated by a super-scripted "C" such as \textsuperscript{GC}. They are obtained by holding down the CTRL key while typing the letter. Control characters are NOT displayed on the TV screen. \textsuperscript{B} and \textsuperscript{C} must be followed by a carriage return. Screen editing characters are indicated by a sub-scripted "E" such as \textsubscript{DE}. 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, \textsuperscript{UC} moves to cursor to right and copies text while \textsuperscript{AE} 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 \textsuperscript{HC}. Forward spaces cursor and copies over written characters. Apple keyboards have H-0 key on right side which also performs this function.</td>
</tr>
<tr>
<td>Control X</td>
<td>Immediately deletes current line.</td>
</tr>
</tbody>
</table>

* If BASIC program is expecting keyboard input, you will have to hit carriage return key after typing control C.
<table>
<thead>
<tr>
<th>CHARACTER</th>
<th>DESCRIPTION OF ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>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>

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.

0 = Black  8 = Brown
1 = Magnenta  9 = Orange
2 = Bark Blue 10 = Grey
3 = Light Purple 11 = Pink
4 = Dark Green 12 = Green
5 = Grey 13 = Yellow
6 = Medium Blue 14 = Blue/Green
7 = Light Blue 15 = White
### 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>C050</td>
<td>10 POKE -16304,0</td>
<td>Set color graphics mode</td>
</tr>
<tr>
<td>C051</td>
<td>20 POKE -16303,0</td>
<td>Set text mode</td>
</tr>
<tr>
<td>C052</td>
<td>30 POKE -16302,0</td>
<td>Clear mixed graphics</td>
</tr>
<tr>
<td>C053</td>
<td>40 POKE -16301,0</td>
<td>Set mixed graphics (4 lines text)</td>
</tr>
<tr>
<td>C054</td>
<td>50 POKE -16300,0</td>
<td>Clear display Page. 2 (BASIC commands use Page 1 only)</td>
</tr>
<tr>
<td>C055</td>
<td>60 POKE -16299,0</td>
<td>Set display to Page 2 (alternate)</td>
</tr>
<tr>
<td>C056</td>
<td>70 POKE -16298,0</td>
<td>Clear HIRES graphics mode</td>
</tr>
<tr>
<td>C057</td>
<td>80 POKE -16297,0</td>
<td>Set HIRES graphics mode</td>
</tr>
<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 WI. W1+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>140</td>
<td>POKE 36,CH</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>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>170</td>
<td>POKE 37,CV</td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>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>200</td>
<td>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_E) 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>(E_F) Clear from cursor to end of line</td>
</tr>
<tr>
<td>FC66</td>
<td>24Ø CALL -922</td>
<td>(JC) Line feed</td>
</tr>
<tr>
<td>FC70</td>
<td>25Ø 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-existant 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 - User specified RAM memory size

XX (HIMEM:) - User Workspace

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

0 0 -
INTRODUCTION

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

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>VN</th>
<th>DSP</th>
<th>NVA</th>
<th>DATA(0)</th>
<th>DATA(1) ...</th>
<th>DATA(n)</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>h₁</td>
<td>h₂</td>
<td>hₙ₊₁</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

*Top of RAM memory is a function of the amount of memory. 16384 will be the value of "HIMEM:" for a 16K system.
The computer keeps track of the amount of memory used for the variable 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 represent 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.
Figure 1

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

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

Figure 2

Figure 3

$800.80C rewritten with labelling
FIGURE 4b

READ/SAVE PROGRAM

<table>
<thead>
<tr>
<th>Statement</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø A=Ø</td>
<td>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.</td>
</tr>
<tr>
<td>1Ø GOTO 1ØØ</td>
<td>This statement moves command to the main program.</td>
</tr>
<tr>
<td>2Ø PRINT &quot;REWIND TAPE THEN START TAPE RECORDER&quot;: INPUT &quot;THEN HIT RETURN&quot;, B$</td>
<td>Lines 20-26 are the write data to tape subroutine.</td>
</tr>
<tr>
<td>22 A=CM-LM: POKE 6Ø,4: POKE 61,8: POKE 62,5: POKE 63,8: CALL -3Ø7</td>
<td>Writing data table to tape</td>
</tr>
<tr>
<td>26 PRINT &quot;DATA TABLE SAVED&quot;: RETURN</td>
<td>Lines 30-38 are the READ data from tape subroutine.</td>
</tr>
<tr>
<td>30 PRINT &quot;REWIND THE TAPE THEN START TAPE RECORDER&quot;: INPUT &quot;AND HIT RETURN&quot;, B$</td>
<td></td>
</tr>
<tr>
<td>32 POKE 6Ø,4: POKE 61,8: POKE 62,5: POKE 63,8: CALL -259</td>
<td></td>
</tr>
<tr>
<td>34 IF A&lt;0 THEN 38: P=LM+A: IF P&gt;HM THEN 38: CM=P: POKE 6Ø, LM MOD 256: POKE 61, LM/256: POKE 62, CM MOD 256: POKE 63, CM/256: CALL -259</td>
<td>Checking the record length (A) for memory requirements if everything is satisfactory the data is READ in.</td>
</tr>
<tr>
<td>36 PRINT &quot;DATA READ IN&quot;: RETURN</td>
<td>Returning control to main program.</td>
</tr>
<tr>
<td>38 PRINT &quot;<em><strong>TOO MUCH DATA BASE</strong></em>&quot;: RETURN</td>
<td></td>
</tr>
</tbody>
</table>

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

Define variable B=0, 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

```
1 >A=1
> Define variable A=-1, then hit RETURN

2 >B=0
> Define variable B=0, then hit RETURN

3 >PRINT PEEK (204) + PEEK (205) * 256
   computer responds with= 2060

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

4 >
* Hit the RESET key, Apple moves into Monitor mode.

5 *800.80C
   Type in VARIABLE TABLE RANGE and HIT the RETURN KEY.

Computer responds with:

0800- Cl 00 86 08 FF FF C2 00
0808 0C 08 00 00 00
```

Example 1
Example 2

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

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

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': GOSUB 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)=0: NEXT I
160 PRINT 'NOW START TAPE RECORDER' :INPUT 'AND THEN HIT RETURN', A$
165 PRINT 'A',A,
170 GOSUB 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
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 (D). Both P and D can range in value from $00 (0) to $FF (255). After the values for frequency and duration are placed into memory a CALL2 statement from BASIC will activate this routine. The speaker is toggled with the machine language statement residing at $02 and then a
delay in time equal to the value in $00$ occurs. This process is repeated until the tone has lasted a relative period of time equal to the duration (value in $01$) and then this program is exited (statement $14$).

Basic Program

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

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

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

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

CONCLUSION

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

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 C00 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 -11699)

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 3895 (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 if 16 will draw the shape rotated 90° 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 $0 byte, it is finished.

The three sections are arranged in a byte like this:

```
0 1 2 3 4 5 6 7
0 0 0 0 P D D D
```

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° 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: \[ \downarrow \downarrow \leftarrow \leftarrow \uparrow \uparrow \uparrow \uparrow \rightarrow \rightarrow \rightarrow \rightarrow \downarrow \downarrow \downarrow \downarrow \]  

Figure 1.

<table>
<thead>
<tr>
<th>C</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>4</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>6</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>8</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>01</td>
</tr>
</tbody>
</table>

Codes:

\[ \uparrow \downarrow \downarrow \leftarrow \rightarrow \rightarrow \rightarrow \rightarrow \downarrow \downarrow \downarrow \downarrow \]  

Figure 2.

<table>
<thead>
<tr>
<th>C</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>001</td>
<td>001</td>
</tr>
<tr>
<td>1</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>2</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>3</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>001</td>
<td>001</td>
</tr>
<tr>
<td>5</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>6</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>8</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>9</td>
<td>111</td>
<td>111</td>
</tr>
</tbody>
</table>

Hexadecimal Codes:

\[ 0000 \rightarrow 0 \]
\[ 0001 \rightarrow 1 \]
\[ 0010 \rightarrow 2 \]
\[ 0011 \rightarrow 3 \]
\[ 0100 \rightarrow 4 \]
\[ 0101 \rightarrow 5 \]
\[ 0110 \rightarrow 6 \]
\[ 0111 \rightarrow 7 \]
\[ 1000 \rightarrow 8 \]
\[ 1001 \rightarrow 9 \]
\[ 1010 \rightarrow A \]
\[ 1011 \rightarrow B \]
\[ 1100 \rightarrow C \]
\[ 1101 \rightarrow D \]
\[ 1110 \rightarrow E \]
\[ 1111 \rightarrow F \]
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 Q=3 TO 50
110 FOR I=1 TO 19
115 FOR J=0 TO 19
120 K=I+J
125 COLOR=(I+3)/(J+3)+I*W/12
130 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
50 REM PONG BY WENKELL BITTER
10 REM /2/77
15 REM PADDLE SWITCHES CONTROL
20 REM PADLE SIZE AFTER A MISS
25 REM OR DURING A HIT
30 OR
35 REM (R)=1, LHJ=1.58 AT R, HLH=1.38 AT R
40 CALL -2351 VSRS 23: : INPUT "HANDS"
45 OR "R" OR "P"? 1,HEP
50 IF (HP=6) THEN 46
55 IF HP=1 THEN 46: IF HP=2 THEN 46
60 OR HEP TO 8 STEP 6
65 OR X=HYP TO 6 AND Y=HYP THEN
70 IF X=1 THEN Y=1: IF Y=12
75 IF X=2 THEN 125: GO TO 250
80 OR X=1, Y=1: GO TO 250
85 OR Y=1, X=1: GO TO 250
90 OR X=1, Y=1: GO TO 250
95 OR X=1, Y=1: GO TO 250
100 FOR T=1 TO 15: X=PEEK (-16336)
105 OR T=1 TO 15: X=PEEK (-16336)
110 OR T=1 TO 15: X=PEEK (-16336)
115 OR T=1 TO 15: X=PEEK (-16336)
120 OR T=1 TO 15: X=PEEK (-16336)
125 OR T=1 TO 15: X=PEEK (-16336)
130 OR T=1 TO 15: X=PEEK (-16336)
135 OR T=1 TO 15: X=PEEK (-16336)
140 OR T=1 TO 15: X=PEEK (-16336)
145 OR T=1 TO 15: X=PEEK (-16336)
150 OR T=1 TO 15: X=PEEK (-16336)
155 OR T=1 TO 15: X=PEEK (-16336)
160 OR T=1 TO 15: X=PEEK (-16336)
165 OR T=1 TO 15: X=PEEK (-16336)
170 OR T=1 TO 15: X=PEEK (-16336)
175 OR T=1 TO 15: X=PEEK (-16336)
180 OR T=1 TO 15: X=PEEK (-16336)
185 OR T=1 TO 15: X=PEEK (-16336)
190 OR T=1 TO 15: X=PEEK (-16336)
195 OR T=1 TO 15: X=PEEK (-16336)
200 OR T=1 TO 15: X=PEEK (-16336)
205 OR T=1 TO 15: X=PEEK (-16336)
210 OR T=1 TO 15: X=PEEK (-16336)
215 OR T=1 TO 15: X=PEEK (-16336)
220 OR T=1 TO 15: X=PEEK (-16336)
225 OR T=1 TO 15: X=PEEK (-16336)
230 OR T=1 TO 15: X=PEEK (-16336)
235 OR T=1 TO 15: X=PEEK (-16336)
240 OR T=1 TO 15: X=PEEK (-16336)
245 OR T=1 TO 15: X=PEEK (-16336)
250 OR T=1 TO 15: X=PEEK (-16336)
255 OR T=1 TO 15: X=PEEK (-16336)
260 OR T=1 TO 15: X=PEEK (-16336)
265 OR T=1 TO 15: X=PEEK (-16336)
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

30 B$="***************
***************": RETURN
35 B$="COLOR SKETCH": RETURN
40 B$="COPYRIGHT APPLE COMPUTER 1977": RETURN
45 B$="THIS PROGRAM ALLOWS YOU TO " : RETURN
50 B$="SKECH COLROD FIGURES IN" : RETURN
55 B$="LOW RESOLUTION GRAPHICS WITH PADDLES": RETURN
60 XK=261:TON=28: GOSUB 85: RETURN
65 XK=10:TOM=10: GOSUB 85: RETURN
70 XK=261:TOM=50: GOSUB 85: XK=30: TOM=29: GOSUB 85: RETURN
95 POKE 1,TON MOD 256: POKE 24 ,TON/256+1: POKE 3,CH: CALL 2: RETURN
96 GOSUB 26: GOSUB 25: PRINT 
100 PRINT : PRINT : GOSUB 78: INPUT
"WHEN READY HIT RETURN",B$ 105 G= 118 B$="A8C8EF411C818C8": CALL -934 115 FOR Z=8 TO 15: COLOR=Z: PLOT Z=2+4,36: VTAB 21: GOSUB 75
Z=2+4,36: VTAB 21: PRINT BK[Z,2,7]; NEXT 2: TAB 1 120 VTAB 22:04="TYPE A LETTER TO CHNG COLOR":GOSUB 25:PRINT 
:"=":TYPE SPACE BAR TO STOP PLOT :"= GOSUB 25:PRINT
125 Y= FNL (X)+30:FNL (Z) = FNL (Y)
>39=SIM:VTAB 24: TAB 1: PRINT 
"CURSOR POSITION X=":Y; " Y=" 
:Y; " ":
130 IF PECK (-16384) THEN 115
135 IF TOM AND 9<1 THEN 150
135 COLOR=2: PLOT X,Y1: IF NOT FLAG THEN 130: COLOR=4: PLOT X,Y
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 BEGIN GAME OF MASTERMIND 8-85-77

10 DIM R(15),A(15),N(15),S(15),K(15)
20 M=10:N=1:T=1:Z=1:W=1
30 P=14:Q=14:R=14:G=14:U=14
40 M=8:K=8:W=8:R=8:N=8:S=8:Z=8

50 REM TEXT: CALL -306: PRINT "WELCOME TO THE GAME OF MASTERMIND!"
60 USE INPUT "YOU OBJECT IS TO GUESS 5 COLOR 5 CASH"
70 IF INPUT "C"=0 THEN 100
80 USE INPUT "I WILL MAKE UP 15 THE 55 NUMBER OF GUESES. THEN"
90 USE INPUT "I WILL GIVE YOU DIFFERENT COLORS TO GUESE FROM."
100 USE INPUT "MORE THAN 15 GUESES--NO!"
110 USE INPUT "EELIGE": PRINT "7 TO 9 GUESE"
120 USE INPUT "S----GOOD": PRINT "10 TO 14 GUESE---AVERAGE"
130 USE INPUT "MORE THAN 14 GUESES--POOR"

140 CALL -304: TAB 7: PRINT "HIT ANY KEY TO BEGIN PLAY"
150 CALL -309: IF PEEK (-16004) THEN 190
160 CALL -309: PEEK -16368.
170 IF Z=1 THEN 180: PRINT "NO MORE COLORS"
180 PRINT "JUMP TO 100"
190 END IF
199 PRINT "";A(1),"";A(2),"";A(3),"";A(4),"";A(5)
200 USE INPUT: PRINT "";A(1),"";A(2),"";A(3),"";A(4),"";A(5)
210 USE INPUT: PRINT "";A(1),"";A(2),"";A(3),"";A(4),"";A(5)
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 3,48: POKE
4,193: POKE 5,165: POKE 6,6
1: POKE 7,32: POKE 8,160: POKE
3,252: POKE 10,160: POKE 11
11: POKE 12,256: POKE 13,14
10 POKE 14,199: POKE 15,24: POKE
16,249: POKE 17,5: POKE 18,
186 POKE 19,11 POKE 20,754
POKE 21,21: POKE 22,6: POKE
23,26 .
15 GOTO 25
20 TI=3: GOSUB 33: RETURN
25 PRINT "*************

*************": RETURN
30 XX=0: TIOM=252: GOSUB 43: RETURN
35 XX=8: TIOM=252: GOSUB 43: RETURN
40 XX=15: TIOM=252: GOSUB 43: RETURN
45 POKE 1,1505: POKE 256: POKE 24,
27: POKE 255+1: POKE 3,48: CALL
2: RETURN
50 ri=(99999999.100))X(1.100)
9(11)(1)(1000000)(1)(1)(1)(1000
3)(11)(1)(1000000)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
195 (3000000)(1)(1)(1)100
50 ri=RI: X(1)(1)(1)(1)(1)(1)(1)(1)
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.
10 PRINT "WELCOME TO THE DRAGON'S MAZE"
20 PRINT "THE FACT THAT THE DRAGON CAN'T GET"
30 PRINT "THERE IT'S COMPLETE, I'LL ERASE."" THE PICTURE. THEN YOU'LL ONLY SEE THE WALLS AS YOU BUMP I"" 40 PRINT "YOU HIT 'X' FOR RIGHT.
50 PRINT "'L' FOR LEFT, 'U' FOR UP,
60 PRINT "D' FOR DOWN. DO NOT HIT RETURN." "
70 PRINT "THE OBJECT IS FOR YOU (THE GREEN GOB)"
80 PRINT "TO GET TO THE DOOR ON THE RIGHT SIDE!"
90 PRINT "BEFORE THE DRAGON (THE RED GOB) EATS!"
100 PRINT "YOU.""
110 PRINT "BEWARE! SOMETIMES THE DRAGON"
120 PRINT "GETS REAL MAD, AND CLIMBS OVER A WALL."
130 PRINT "BUT MOST OF THE TIME, HE CAN'T Go OVER."
140 PRINT "WHO HAS TO GO KILLING."
150 PRINT "YOU CAN OFTEN TELL WHERE A WALL:"}

200 PRINT "IS, EVEN BEFORE YOU CAN SEE IT, BY"
210 PRINT "THE OBSTACLE IS FOR YOU (THE GREEN GOB)"
220 PRINT "TO GET TO THE DOOR ON THE RIGHT SIDE!"
230 PRINT "BEFORE THE DRAGON (THE RED GOB) EATS!"
240 PRINT "YOU."
250 PRINT "BEWARE! SOMETIMES THE DRAGON"
260 PRINT "GETS REAL MAD, AND CLIMBS OVER A WALL."
270 PRINT "BUT MOST OF THE TIME, HE CAN'T Go OVER."
280 PRINT "WHO HAS TO GO KILLING."
290 PRINT "YOU CAN OFTEN TELL WHERE A WALL:"
DRAGON MAZE cont.

1225 XR=3*XR-2;HT=3*Y-2
1230 HY= RND (13)+1
1240 COLOR=8; VLINE 3*W-2,3*W-1
   AT 35
1250 SX=13;SY=W
1260 GX=3*W-2;GY=3*SY-2
1270 KD=1
1500 K=PEEK (-15064); IF K<128 THEN
   1510 LOAD -16368,8
1515 DX=K: GDOSUB 7000:K=00
1520 IF SY=K AND SX=SY THEN 3000
1525 IF K= ASC('R') THEN 2000
1530 IF K=ASC('L') THEN 2000
1540 IF K=ASC('U') THEN 3000
1550 IF K=ASC('D') THEN 3500
1560 GDOSUB 8800:GOTO 1500
2000 DX=11;DY=8
2010 IF MX+13*(KY-1) MOD 16 THEN
   2020 GOTO 2020
2025 DX=0: DIY=-1
2030 IF MX+13*(Y-2)) MOD 4000 THEN 4000
2040 IF K<15A THEN 7000: IF Y<SY THEN
   2050 7000
2060 IF K>SY THEN 7140: IF Y>SY THEN
   7150 7140
2070 IF SX<13 THEN 7050: IF T(SX+
   13*(SY-1))=0 THEN 7140: IF
2080 MX+13*(SY-1) MOD 16 THEN
   7200
2090 DX=1;DY=6
2095 COLOR=8
2100 PX=3*SY-2;RY=3*SY-2
2105 FOR I=1 TO 3:KG=PIX:RY=RY+1
   2110 PRINT 'Y'
2110 COLOR=0
2120 FOR K=8 TO 1: FOR L=0 TO 1:
   2125 PLOT KG,SY; NEXT L,L; COLOR=
   2130 800: FOR K=8 TO 1: FOR L=0 TO
   2140 1: PLOT KG,KY; NEXT L,L;
   2145 KG=KG+1:SY=SY+1
2150 NEXT I
2150 7050 IF SY<13 THEN 7100: IF T(SX+
   13*(SY-1))=0 THEN 7100: IF
2160 RETURN
2160 2020
2165 IF X<13 AND Y<SY THEN 5000
2170 GOTO 1500
2180 DX=-1;DY=0
2185 IF MX+13*(Y-1)-1 MOD 16 THEN
   2190 NEXT I
2195 IF K=3*YL-2
2200 PRINT "YOU WUK!
" 2210 PRINT "SCORE=";S+3
2220 END
DRAGON MAZE cont.

7100 DX=-1;DY=0: GOTO 7000
7150 IF SY=1 THEN 7005: IF T(SX+
13*(SY-1))>0 THEN 7160: IF
M(SX+13*(SY-1)-13)/10 THEN
7005
7160 DX=0;DY=-1: GOTO 7000
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 C^C (control C). NOTE: "adrs" is a four digit hexadecimal number and "data" is a two digit hexadecimal number. Remember to press "return" button at the end of each line.

<table>
<thead>
<tr>
<th>Command Format</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examine Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrs</td>
<td>*C0F2</td>
<td>Examines (displays) single memory location of (adrs)</td>
</tr>
<tr>
<td>adrs1.adrs2</td>
<td>*1024.1048</td>
<td>Examines (displays) range of memory from (adrs1) thru (adrs2)</td>
</tr>
<tr>
<td>(return)</td>
<td>*(return)</td>
<td>Examines (displays) next 8 memory locations.</td>
</tr>
<tr>
<td>.adrs2</td>
<td>*.4096</td>
<td>Examines (displays) memory from current location through location (adrs2)</td>
</tr>
<tr>
<td>Change Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrs:data 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. adrs3M</td>
<td>*100&lt;B010.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>Verify Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrs1&lt;adrs2 adrs3V</td>
<td>*100&lt;B010.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</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>$(monitor command)</td>
<td>!$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>adrs:(65Ø2 MNEMONIC instruction)</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 instruction)</td>
<td>! STA ØlFF</td>
<td>Assembles instruction into next available memory location. (Note space between &quot;f&quot; and instruction)</td>
</tr>
<tr>
<td>(TURN-OFF)</td>
<td>! (Reset Button)</td>
<td>Exits mini-assembler and returns to system monitor.</td>
</tr>
</tbody>
</table>

**Monitor Program Execution and Debugging**

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

**Note 1:**

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

Example:

\[
\begin{align*}
A &= 3C \\
X &= FF \\
Y &= 00 \\
P &= 32 \\
S &= F2 \\
*: FF & \text{ Changes A register only} \\
*: FF 00 33 & \text{ Changes A, X, and Y registers}
\end{align*}
\]

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

**Hexadecimal Arithmetic**

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

**Note 2:**

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

**Multiple Commands**

*100L 400G AFFT  Multiple monitor commands may be given on same line if separated by a "space".  

*LLLLL  Single letter commands may be repeated without spaces.
SPECIAL CONTROL AND EDITING CHARACTERS

"Control" characters are indicated by a super-scripted "C" such as GC. They are obtained by holding down the CTRL key while typing the specified letter. Control characters are NOT displayed on the TV screen. BC and CC must be followed by a carriage return. Screen editing characters are indicated by a sub-scripted "E" such as DE. 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, UC moves to cursor to right and copies text while AE 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><strong>RESET key</strong></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><strong>Control B</strong></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><strong>Control C</strong></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><strong>Control G</strong></td>
<td>Sounds bell (beeps speaker)</td>
</tr>
<tr>
<td><strong>Control H</strong></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><strong>Control J</strong></td>
<td>Issues line feed only</td>
</tr>
<tr>
<td><strong>Control V</strong></td>
<td>Compliment to H(^C). Forward spaces cursor and copies over written characters. Apple keyboards have &quot;+&quot; key on right side which also performs this function.</td>
</tr>
<tr>
<td><strong>Control X</strong></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>AE</td>
<td>Move cursor to right</td>
</tr>
<tr>
<td>BE</td>
<td>Move cursor to left</td>
</tr>
<tr>
<td>CE</td>
<td>Move cursor down</td>
</tr>
<tr>
<td>DE</td>
<td>Move cursor up</td>
</tr>
<tr>
<td>EE</td>
<td>Clear text from cursor to end of line</td>
</tr>
<tr>
<td>FE</td>
<td>Clear text from cursor to end of page</td>
</tr>
<tr>
<td>@E</td>
<td>Home cursor to top of page, clear text to end of page.</td>
</tr>
</tbody>
</table>
## Special Controls and Features

<table>
<thead>
<tr>
<th>Hex</th>
<th>BASIC Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><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_E) 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>(EE) Clear from cursor to end of line</td>
</tr>
<tr>
<td>FC66</td>
<td>24Ø CALL -922</td>
<td>(JC) Line feed</td>
</tr>
<tr>
<td>FC7Ø</td>
<td>25Ø 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>C03Ø</td>
<td>36Ø X=PEEK(-16336)</td>
<td>Toggle speaker</td>
</tr>
<tr>
<td></td>
<td>365 POKE -16336,Ø</td>
<td></td>
</tr>
<tr>
<td>C000</td>
<td>37Ø X=PEEK(-16384)</td>
<td>Read keyboard; if X&gt;127 then key was pressed.</td>
</tr>
<tr>
<td>C01Ø</td>
<td>38Ø POKE -16368,Ø</td>
<td>Clear keyboard strobe - always after reading keyboard.</td>
</tr>
<tr>
<td>C061</td>
<td>39Ø 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>40Ø X=PEEK(-16286)</td>
<td>Read PDL(1) push button switch.</td>
</tr>
<tr>
<td>C063</td>
<td>41Ø X=PEEK(-16285)</td>
<td>Read PDL(2) push button switch.</td>
</tr>
<tr>
<td>C058</td>
<td>42Ø POKE -16296,Ø</td>
<td>Clear Game I/O AN0 output</td>
</tr>
<tr>
<td>C059</td>
<td>43Ø POKE -16295,Ø</td>
<td>Set Game I/O AN0 output</td>
</tr>
<tr>
<td>C05A</td>
<td>44Ø POKE -16294,Ø</td>
<td>Clear Game I/O AN1 output</td>
</tr>
<tr>
<td>C05B</td>
<td>45Ø POKE -16293,Ø</td>
<td>Set Game I/O AN1 output</td>
</tr>
<tr>
<td>C05C</td>
<td>46Ø POKE -16292,Ø</td>
<td>Clear Game I/O AN2 output</td>
</tr>
<tr>
<td>C05D</td>
<td>47Ø POKE -16291,Ø</td>
<td>Set Game I/O AN2 output</td>
</tr>
<tr>
<td>C05E</td>
<td>48Ø POKE -16290,Ø</td>
<td>Clear Game I/O AN3 output</td>
</tr>
<tr>
<td>C05F</td>
<td>49Ø POKE -16289,Ø</td>
<td>Set Game I/O AN3 output</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>ADDRESS</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>LOC0</td>
<td>EPZ $00</td>
<td></td>
</tr>
<tr>
<td>LOC1</td>
<td>EPZ $01</td>
<td></td>
</tr>
<tr>
<td>WNDLFT</td>
<td>EPZ $20</td>
<td></td>
</tr>
<tr>
<td>WNDWDT</td>
<td>EPZ $21</td>
<td></td>
</tr>
<tr>
<td>WNDTOP</td>
<td>EPZ $22</td>
<td></td>
</tr>
<tr>
<td>WNDBTM</td>
<td>EPZ $23</td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>EPZ $24</td>
<td></td>
</tr>
<tr>
<td>CV</td>
<td>EPZ $25</td>
<td></td>
</tr>
<tr>
<td>GBASL</td>
<td>EPZ $26</td>
<td></td>
</tr>
<tr>
<td>GBASH</td>
<td>EPZ $27</td>
<td></td>
</tr>
<tr>
<td>BASL</td>
<td>EPZ $28</td>
<td></td>
</tr>
<tr>
<td>BASH</td>
<td>EPZ $29</td>
<td></td>
</tr>
<tr>
<td>BAS2L</td>
<td>EPZ $2A</td>
<td></td>
</tr>
<tr>
<td>BAS2H</td>
<td>EPZ $2B</td>
<td></td>
</tr>
<tr>
<td>H2</td>
<td>EPZ $2C</td>
<td></td>
</tr>
<tr>
<td>LMMNM</td>
<td>EPZ $2C</td>
<td></td>
</tr>
<tr>
<td>KTNL</td>
<td>EPZ $2C</td>
<td></td>
</tr>
<tr>
<td>V2</td>
<td>EPZ $2D</td>
<td></td>
</tr>
<tr>
<td>RMBNM</td>
<td>EPZ $2D</td>
<td></td>
</tr>
<tr>
<td>FTNH</td>
<td>EPZ $2D</td>
<td></td>
</tr>
<tr>
<td>MASK</td>
<td>EPZ $2E</td>
<td></td>
</tr>
<tr>
<td>CHKSUM</td>
<td>EPZ $2E</td>
<td></td>
</tr>
<tr>
<td>FORMAT</td>
<td>EPZ $2E</td>
<td></td>
</tr>
<tr>
<td>LASTIN</td>
<td>EPZ $2F</td>
<td></td>
</tr>
<tr>
<td>LENGTH</td>
<td>EPZ $2F</td>
<td></td>
</tr>
<tr>
<td>SIGN</td>
<td>EPZ $2F</td>
<td></td>
</tr>
<tr>
<td>COLOR</td>
<td>EPZ $30</td>
<td></td>
</tr>
<tr>
<td>MODE</td>
<td>EPZ $31</td>
<td></td>
</tr>
<tr>
<td>INVFNG</td>
<td>EPZ $32</td>
<td></td>
</tr>
<tr>
<td>PROMPT</td>
<td>EPZ $33</td>
<td></td>
</tr>
<tr>
<td>YSAV</td>
<td>EPZ $34</td>
<td></td>
</tr>
<tr>
<td>YSAV1</td>
<td>EPZ $35</td>
<td></td>
</tr>
<tr>
<td>CSWL</td>
<td>EPZ $36</td>
<td></td>
</tr>
<tr>
<td>CSWH</td>
<td>EPZ $37</td>
<td></td>
</tr>
<tr>
<td>KSWL</td>
<td>EPZ $38</td>
<td></td>
</tr>
<tr>
<td>KSWH</td>
<td>EPZ $39</td>
<td></td>
</tr>
<tr>
<td>PCL</td>
<td>EPZ $3A</td>
<td></td>
</tr>
<tr>
<td>PCH</td>
<td>EPZ $3B</td>
<td></td>
</tr>
<tr>
<td>XQT</td>
<td>EPZ $3C</td>
<td></td>
</tr>
<tr>
<td>A1L</td>
<td>EPZ $3C</td>
<td></td>
</tr>
<tr>
<td>A1H</td>
<td>EPZ $3D</td>
<td></td>
</tr>
<tr>
<td>A2L</td>
<td>EPZ $3E</td>
<td></td>
</tr>
<tr>
<td>A2H</td>
<td>EPZ $3F</td>
<td></td>
</tr>
<tr>
<td>A3L</td>
<td>EPZ $40</td>
<td></td>
</tr>
<tr>
<td>A3H</td>
<td>EPZ $41</td>
<td></td>
</tr>
<tr>
<td>A4L</td>
<td>EPZ $42</td>
<td></td>
</tr>
<tr>
<td>A4H</td>
<td>EPZ $43</td>
<td></td>
</tr>
<tr>
<td>A5L</td>
<td>EPZ $44</td>
<td></td>
</tr>
<tr>
<td>A5H</td>
<td>EPZ $45</td>
<td></td>
</tr>
</tbody>
</table>
ORG $F800  ROM START ADDRESS

F800: 4A  PLOT  LSB  Y-COORD/2
F801: 08  PHP  SAVE LSB IN CARRY
F802: 20 47 F8  JSR  GBASCALC  CALC BASE ADR IN GBASL,H
F805: 28  PLP  RESTORE LSB FROM CARRY
F806: A9 0F  LDA  #$0F  MASK $0F IF EVEN
F808: 90 02  BCC  RTMASK
F80A: 69 E0  ADC  #$E0  MASK $F0 IF ODD
F80E: B1 26  RTMASK  STA  MASK
F810: 45 30  EOR  COLOR  EOR COLOR
F812: 25 2E  AND  MASK  AND MASK
F814: 51 26  EOR  (GBASL),Y  XOR DATA
F816: 85 2E  RTMASK  STA  (GBASL),Y  TO DATA
F818: 60  RTS
F819: 20 00 F8  HLINE  JSR  PLOT  PLOT SQUARE
F81C: C4 2C  HLINE1  CPY  H2  DONE?
F81E: B0 11  BCS  RTS1  YES, RETURN
F820: C8  INY  NO, INC INDEX (X-COORD)
F822: 49 F8  JSR  PLOT1  PLOT NEXT SQUARE
F824: 90 F6  BCC  HLINE1  ALWAYS TAKEN
F826: 69 01  VLINEZ  ADC  #$01  NEXT Y-COORD
F828: 48  VLINE  PHA  SAVE ON STACK
F829: 20 00 F8  JSR  PLOT  PLOT SQUARE
F82C: 68  PLA
F82D: C5 2D  CMP  V2  DONE?
F82F: 90 F5  BCC  VLINEZ  NO, LOOP
F831: 60  RTS1  RTS
F832: A0 2F  CLRSCR  LDY  #$2F  MAX Y, FULL SCRNL CLR
F834: D0 02  BNE  CLRSC2  ALWAYS TAKEN
F836: A0 27  CLETOP  LDY  #$27  MAX Y, TOP SCREEN CLR
F838: 84 2D  CLRSC2  STY  V2  STORE AS BOTTOM COORD
F83A: A0 27  LDY  #$27  RIGHTMOST X-COORD (COLUMN)
F83C: A9 00  CLRSC3  LDA  #$00  TOP COORD FOR VLINE CALLS
F83E: 85 30  STA  COLOR  CLEAR COLOR (BLACK)
F840: 20 28 F8  JSR  VLINE  DRAW VLINE
F843: 88  DEY  NEXT LEFTMOST X-COORD
F844: 10 F6  BPL  CLRSC3  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
F858: 0A              ASL   A
F859: 0A              ASL   A
F85A: 05 26          ORA   GBASL
F85C: 85 26          STA   GBASL
F85E: 60              RTS
F85F: A5 30          NXTCOL   LDA   COLOR   INCREMENT COLOR BY 3
F861: 18              CLC
F862: 69 03          ADC   #$03
F864: 29 0F          SETCOL   AND   #$0F   SETS COLOR=$F-A MOD 16
F866: 85 30          STA   COLOR
F868: 0A              ASL   A   BOTH HALF BYTES OF COLOR EQUAL
F869: 0A              ASL   A
F86A: 0A              ASL   A
F86B: 0A              ASL   A
F86C: 05 30          ORA   COLOR
F86E: 85 30          STA   COLOR
F870: 60              RTS
F871: 4A              SCRN   LSR   A   READ SCREEN Y-COORD/2
F872: 08              PHP   SAVE LSB (CARRY)
F873: 20 47 F8      JSR   GBASCALC   CALC BASE ADDRESS
F876: B1 26          LDA   (GBASL),Y  GET BYTE
F878: 28              PLP   RESTORE LSB FROM CARRY
F879: 90 04          SCRN2   LDA   COLOR      INCREMENT COLOR BY 3
F87B: 4A              LSR   A
F87C: 4A              LSR   A
F87D: 4A              LSR   A   SHIFT HIGH HALF BYTE DOWN
F87E: 4A              LSR   A
F87F: 29 0F          RTMSKX   AND   #$0F   MASK 4-BITS
F881: 60              RTS
F882: A6 3A          INSDS1   LDX   PCL        PRINT PCL,H
F884: A4 3B          LDY   PCH
F886: 20 96 FD      JSR   PRYX2
F889: 20 48 F9      JSR   PRBLNK   FOLLOWED BY A BLANK
F88C: A1 3A          LDA   (PCL,X)    GET OP CODE
F88E: A8              INSDS2   TAY
F890: 90 09          BCC   IEVEN   IF EVEN, USE LO H
F892: 6A              ROG   BIT 1 TEST
F893: B0 10          BCS   ERR   XXXXXX11 INVALID OP
F895: C9 A2          CMP   #$A2
F897: F0 0C          BEQ   ERR   OPCODE $89 INVALID
F899: 29 87          AND   #$87   MASK BITS
F89B: 4A              IEVEN   LSR   A   LSB INTO CARRY FOR L/R TEST
F89C: AA              TAX
F89D: BD 62 F9      LDA   FMT1,X   GET FORMAT INDEX BYTE
F8A0: 20 79 F8      JSR   SCRNCALC   R/L H-BYTE ON CARRY
F8A3: D0 04          BNE   GETFMT
F8A5: A0 80          ERR   LDY   #$80   SUBSTITUTE $80 FOR INVALID OPS
F8A7: A9 00          LDA   #$00   SET PRINT FORMAT INDEX TO 0
F8A9: AA        GETFMT   TAX
F8AA: BD A6 F9      LDA   FMT2,X   INDEX INTO PRINT FORMAT TABLE
F8AD: 85 2E          STA   FORMAT   SAVE FOR ADR FIELD FORMATTING
F8AF: 29 03          AND   #$03   MASK FOR 2-BIT LENGTH
F8B1: 85 2F          STA   LENGTH
F8B3: 98              TYA   OPCODE
F8B4: 29 8F          AND   #$8F   MASK FOR 1XXX1010 TEST
F8B6: AA              TAX   SAVE IT
F8B7: 98              TYA   OPCODE TO A AGAIN
F8B8: A0 03          LDY   #$03
F8BA: E0 8A          CPX   #$8A
F8BB: F0 0B          BEQ   MNNDX3
F8BE: 4A              MNNDX1   LSR   A
F8BF: 90 0B          BCC   MNNDX3   FORM INDEX INTO MNEMONIC TABLE
F8C1: 4A              LSR   A
F8C2: 4A              MNNDX2   LSR   A   1) 1XXX1010-#001011XXX
F8C3: 09 20          ORA   #$20   2) XXXXXX01-#001111XXX
F8C5: 88              DEY   3) XXXYXX-#00110XXX
F8C6: D0 FA          BNE   MNNDX2   4) XXXYY10-#00100XXX
F8C8: C8              INY   5) XXXXX00-#000XXX
F8CA: 8B              MNNDX3   DEY
F8CC: D0 F2          BNE   MNNDX1
F8CD: 60              RTS
F8DD: FF FF FF      DFB   SFF,SFF,SFF
F8D0: 20 82 F8      INSTDSP   JSR   INSDS1   GEN FMT, LEN BYTES
F8D3: 48              PHA   SAVE MNEMONIC TABLE INDEX
F8D4: B1 3A          PRINTOP   LDA   (PCL),Y
F8D6: 20 DA FD      JSR   PRBYTE
F8D9: A2 01          LDX   #$01   PRINT 2 BLANKS
F8DB: 20 4A F9      PRINTBL   JSR   PRBL2
F8DE: C4 2F          CPY   LENGTH   PRINT INST (1-3 BYTES)
F8E0: C8              INY   IN A 12 CHR FIELD
F8E1: 90 F1          BCC   PRINTOP
F8E3: A2 03          LDX   #$03
F8E5: C0 04          CPY   #$04
78
F8E7: 90 F2              BCC  PRNTBL
F8E9: 68              PLA              RECOVER MNEMONIC INDEX
F8EA: A8              TAY
F8EB: B9 C0 F9              LDA MNEML,Y
F8EC: 85 2C              STA LMNEM FETCH 3-CHAR MNEMONIC
F8F0: B9 00 FA              LDA MNEML,Y (PACKED IN 2-BYTES)
F8F3: 85 2D              STA RMMEM
F8F5: A9 00 PRMN1              LDA #$00
F8F7: A0 05              LDY #$05
F8F9: 06 2D PRMN2              ASL RMMEM SHIFT 5 BITS OF
F8FB: 26 2C              ROL LMNEM (CLEAR CARRY)
F8FD: 88              DEY
F8FE: D0 F8              BNE PRMN2
F901: 69 BF              ADC #$BF ADD "?" OFFSET
F903: 20 ED FD              JSR COUT OUTPUT A CHAR OF MNEM
F906: CA              DEX
F907: D0 EC              BNE PRMN1
F909: 20 48 F9              JSR PRBLNK OUTPUT 3 BLANKS
F90C: A4 2F              STA RMMEM
F90D: A9 00     PRMN1              LDA #$00
F90E: A0 05              LDY #$05
F910: 06 2D     PRMN2              ASL FORMAT
F911: 90 0E              BCC PRADR3
F913: BD B3 F9              LDA CHAR1-1,X
F916: 20 ED FD              JSR COUT
F918: BD B9 F9              JSR PRADR1 ADDR,
F91B: A4 2F     PRADR2              STA FORMAT
F91C: C9 E8              CMP #$E8 HANDLE REL ADDR MODE
F91D: B1 3A              LDA (PCL),Y SPECIAL (PRINT TARGET,
F91E: 90 F2              BCC PRADR4 NOT OFFSET
F920: 20 56 F9 RELADR              JSR PCADJ3
F922: AA              TAX
F923: E8              INC
F925: D0 01              BNE PRNTX +1 TO Y,X
F927: D0 EC              BNE PRADR1
F929: 60              RTS
F92A: 88              DEY
F92B: 30 E7              BMI PRADR2
F92D: 20 DA FD              JSR PRBYTE
F92F: A5 2E PRADR5              CMP #$2E HANDLE REL ADDR MODE
F932: C9 E8              CMP #$E8
F934: B1 3A              LDA (PCL),Y SPECIAL (PRINT TARGET,
F936: 90 F2              BCC PRADR4 NOT OFFSET
F938: 20 56 F9 RELADR              JSR PCADJ3
F93A: AA              TAX
F93B: E8              INC
F93D: D0 01              BNE PRNTX +1 TO Y,X
F93F: C8              INY
F940: 98              PRNTX TTA
F942: 20 DA FD              JSR PRBYTE OUTPUT TARGET ADDR
F945: 4C DA FD              JMP PRBYTE
F948: A2 03 PRBLNK              JMP #$03 BLANK COUNT
F94A: A9 A0 PRBL2              JMP #$A0 LOAD A SPACE
F94C: 20 ED FD PRBL3              JSR COUT OUTPUT A BLANK
F94E: CA              DEX
F950: D0 F8              BNE PRBL2 LOOP UNTIL COUNT=0
F952: 60              RTS
F953: 3B              SEC
F954: A5 2F              CMP #$2E 0=1-BYTE, 1=2-BYTE
F956: A4 3B              CMP #$E8 2=3-BYTE
F958: AA              TAX TEST DISPLACEMENT SIGN
F95A: 10 01 BPL PCADJ4 (FOR REL BRANCH)
F95B: 88              DEX EXTEND NBG BY DEC PCH
F95C: 65 3A PCADJ4              ADC PCL
F95E: 90 01              BCC RTS2 PCL+LENGTH(CR DISPL)+1 TO A
F960: C8              INY CARRY INTO Y (PCH)
F961: 60 RTS2 RTS
F962: 04 20 54
F965: 30 0D FMT1              DFB $04,$20,$54,$30,$0D
F967: 80 04 90
F96A: 03 22              DFB $80,$04,$90,$50,$32
F96C: 54 33 0D              DFB $54,$33,$50,$50,$0D
F96F: 80 04              DFB $54,$33,$50,$50,$0D
F971: 90 04 20
F974: 54 33              DFB $90,$04,$20,$54,$33
F976: OD 80 04
F979: 90 04              DFB $90,$80,$50,$50,$0D
F97B: 20 54 3B
F97E: OD 80
F980: 04 90 00
F983: 22 44              DFB $04,$90,$50,$22,$44
F985: 33 0D C8
F988: 44 00              DFB $33,$0D,$58,$44,$0D

79
FB81: 20 A4 FB DIVPM JSR MD1 ABS VAL OF AC, AUX.

FB84: A0 10 DIV LDY #$10 INDEX FOR 16 BITS

FB86: 06 50 DIV2 ASL ACL

FB88: 26 51 ROL ACN ROL XTNDL XTND/AUX

FB8A: 26 52 ROL XTNDH TO AC.

FB8E: 38 SEC

FB91: E5 54 SBC AUXL MOD TO XTND.

FB93: AA TAX

FB94: A5 53 LDA XTNDH

FB96: E5 55 SBC AUXH

FB98: 90 06 BCC DIV3

FB9A: 86 52 STX XTNDL

FB9C: 85 53 STA XTNDH

FB9E: E6 50 INC ACL

FB9F: A2 54 LDX #ACL

FBAD: A0 00 MD1 LDY #$00 ABS VAL OF AC, AUX

FBAA: 8F 2F STY SIGN WITH RESULT SIGN

FBAB: A2 54 LDX #AUXL IN LSB OF SIGN.

FBAC: 26 50 ROL ACL

FBAD: A2 50 LDX #ACL

FBAF: B5 01 MD3 LDA LOCI,X X SPECIFIES AC OR AUX

FBAD: 85 06 BCC DIV3

FB9F: 86 52 STX XTNDL

FB9C: 85 53 STA XTNDH

FB9E: E6 50 INC ACL

FB9A: 10 D0 BPL MDRTS

FB9B: 38 SEC

FB9D: 00 0F STA LOC0,X IF NEG.

FB9F: 85 01 STA LOC1,X

FBB1: 10 0D BNE MDRTS

FBB4: 98 TYA

FBB5: F5 00 SBC LOC0,X COMPL SPECIFIED REG

FBB7: 95 00 STA LOC0,X IF NEG.

FBB9: 98 TYA

FBBA: F5 01 SBC LOC1,X

FBBB: 95 01 STA LOC1,X

FBBE: E6 2F INC SIGN

F0C0: 60 MDRTS RTS

F0C1: 48 BASCALC PHA CALC BASE ADR IN BASL,H

F0C2: 4A LSR A FOR GIVEN LINE NO

F0C3: 29 03 AND #$03 0&1t-LINE NO.&1t-#$17

F0C5: 09 04 ORA #$04 ARG=000ABCD, GENERATE

F0C7: 85 29 STA BASH BASH=000000CD

F0C9: 68 PLA AND

F0CA: 29 18 AND #$18 BASL=EABAB000

F0CC: 90 02 BCC BSCLC2

F0CE: 69 7F ADC #$7F

FD0D: 85 2B BSCLC2 STA BASL

FD0E: 38 SEC

FD0F: 0A PLL MDRTS

FD10: 05 28 ORA BASL

FD11: 85 28 STA BASL

FD12: 60 RTS

FD13: C9 87 BELL1 CMP #$87 BELL CHAR? (CNTRL-G)

FD15: 00 40 STA #$40 DELAY .01 SECONDS

FD17: 20 A8 FC JSR WAIT

FD18: A0 CO LDY #$C0

FD19: A9 0C BELL2 LDA #$0C TOGGLE SPEAKER AT

FD1A: 85 28 BSCLC2 STA BASL

FD1B: 38 SEC

FD1C: 0A PLL

FD1D: 05 28 ORA BASL

FD1E: 85 28 STA BASL

FD1F: 60 RTS

FD20: C9 87 BELL1 CMP #$87 BELL CHAR? (CNTRL-G)

FD21: D0 12 BNE RTS2B NO, RETURN

FD23: A9 40 LDA #$40 DELAY .01 SECONDS

FD25: 20 A8 FC JSR WAIT

FD26: A0 CO LDY #$C0

FD27: A9 0C BELL2 LDA #$0C TOGGLE SPEAKER AT

FD28: 20 A8 FC JSR WAIT 1 KHZ FOR .1 SEC.

FD29: AD 30 CO LDA SPKR

FD2A: 7F ADC #$7F

FD2B: 60 RTS2B RTS

FD2C: C9 87 BELL1 CMP #$87 BELL CHAR? (CNTRL-G)

FD2D: D0 12 BNE RTS2B NO, RETURN

FD2F: 85 06 BCC DIV3

FD30: 09 04 ORA #$04 ARG=000ABCD, GENERATE

FD32: 85 29 STA BASH BASH=000000CD

FD34: 68 PLA AND

FD35: 29 18 AND #$18 BASL=EABAB000

FD37: 90 02 BCC BSCLC2

FD38: 69 7F ADC #$7F

FD39: 85 2B BSCLC2 STA BASL

FD3A: 38 SEC

FD3B: 0A PLL MDRTS

FD3C: 05 28 ORA BASL

FD3D: 85 28 STA BASL

FD3E: 60 RTS

FD3F: C9 87 BELL1 CMP #$87 BELL CHAR? (CNTRL-G)

FD40: D0 12 BNE RTS2B NO, RETURN

FD42: A9 40 LDA #$40 DELAY .01 SECONDS

FD44: 20 A8 FC JSR WAIT

FD45: A0 CO LDY #$C0

FD46: A9 0C BELL2 LDA #$0C TOGGLE SPEAKER AT

FD47: 20 A8 FC JSR WAIT 1 KHZ FOR .1 SEC.

FD48: AD 30 CO LDA SPKR

FD49: 7F ADC #$7F

FD4A: 60 RTS2B RTS

FD4B: A4 24 STOADV LDY CH CURSOR H INDEX TO Y-REG

FD4C: 91 28 STA (BASL),Y STORE CHAR IN LINE

FD4D: E6 24 ADVANCE INC CH INCREMENT CURSOR H INDEX

FD4E: A5 24 LDA CH (MOVE RIGHT)

FD4F: C5 21 CMP MWNDTH BEYOND WINDOW WIDTH?

FD50: B0 66 BCS CR YES CR TO NEXT LINE

FD51: 60 RTS3 RTS NO,RETURN

FD52: C9 A0 VIDOUT CMP #$A0 CONTROL CHAR?

FD53: B0 EF BCS STOADV NO,OUTPUT IT.

FD54: 00 40 STA #$40 DELAY .01 SECONDS

FD55: 20 A8 FC JSR WAIT

FD56: A0 CO LDY #$C0

FD57: 85 28 STA BASL

FD58: 60 RTS

FD59: C9 8A CMP #$8A LINE FEED?

FD5A: F0 5A BEQ CR YES.

FD5B: C9 8A CMP #$8A LINE FEED?

FD5C: 9A 8A CMP #$8A LINE FEED?

FD5D: 9A 8A CMP #$8A LINE FEED?

FD5E: 9A 8A CMP #$8A LINE FEED?

FD5F: 9A 8A CMP #$8A LINE FEED?
FC1E: B0 0B
     BCS RTS4  IF TOP LINE THEN RETURN
FC20: C6 25
     DECB CV  DBC CURSOR V-INDEX
FC22: A5 25
     VTABZ LDA CV  GET CURSOR V-INDEX
FC24: 20 C1
     VTABZ JSR BASCALC  GENERATE BASE ADR
FC27: 65 20
     ADC WNDLFT  ADD WINDOW LEFT INDEX
FC29: 85 28
     STA BASL  TO BASL
FC2B: 60 08
     RTS4 RTSB
FC2C: 49 0C
     ESC1 BOR #$C0 ESC?
FC2E: F0 28
     BEQ HOME  IF SO, DO HOME AND CLEAR
FC30: 69 0D
     ADC #$F0  ESC-A OR B CHECK
FC32: 90 C5
     BCC CLREOL  E, CLEAR TO END OF LINE
FC34: F0 DA
     BEQ BS  B, BACKSPACE
FC36: 69 FD
     ADC #$FD  ESC-C OR D CHECK
FC38: 90 2C
     BCC LF  C, DOWN
FC3A: F0 DE
     BEQ UP  D, GO UP
FC3C: 69 FD
     ADC #$FD  ESC-E OR F CHECK
FC3E: 90 5C
     BCC CLREOL  E, CLEAR TO END OF LINE
FC40: D0 E9
     BNE RTS4  NOT F, RETURN
FC42: A4 24
     CLREOP LDY CH  CURSOR H TO Y INDEX
FC44: A5 28
     SCRL1 LDA BASL  COPY BASL,H
FC46: 85 2A
     STA BAS2L  TO BAS2L,H
FC48: A5 29
     LDA BASH
FC4A: 85 2B
     STA BAS2H
FC4C: A4 21
     LDY WNDWDTH  INIT Y TO RIGHTMOST INDEX
FC4E: 88
     DEY  OF SCROLLING WINDOW
FC50: 68
     PLA
FC52: 69 DF
     ADC #$F0  (CARRY IS SET)
FC54: 90 F0
     BNE CLREOL  NO, KEEP CLEARING LINES
FC56: B0 CA
     BS LDA CV  CURSOR V TO A-REGISTER
FC58: A5 22
     HOME LDA WNDTOP  INIT CURSOR V
FC5A: 85 25
     STA CV  AND H-INDICES
FC5C: A0 00
     LDY #$00  THEN CLEAR TO END OF PAGE
FC5E: 84 24
     STY CH  THEN CLEAR TO END OF PAGE
FC60: F0 E4
     BEQ CLEOL1  CLEAR FROM H INDEX=0 FOR REST
FC62: A9 00
     CR LDA #$00  CURSOR TO LEFT OF INDEX
FC64: 85 24
     STA CH  (RET CURSOR H=0)
FC66: E6 25
     LF INC CV  INCR CURSOR V(DOWN 1 LINE)
FC68: A5 25
     LDA CV
FC6A: C5 23
     CMP WNDBTM  DONE TO BOTTOM OF WINDOW?
FC6C: 90 F0
     BCC VTABZ  NO, SET BASE ADDR
FC6E: C6 25
     DDC CV  DECK CURSOR V (BACK TO BOTTOM)
FC70: A5 22
     SCROLL LDA WNDTOP  START AT TOP OF SCRL WND
FC72: 48
     PHA
FC74: 20 24
     FC JSR VTABZ  GENERATE BASE ADR
FC76: A5 28
     SCRL1 LDA BASL  COPY BASL,H
FC78: 85 2A
     STA BAS2L  TO BAS2L,H
FC7A: A5 29
     LDA BASL
FC7C: 85 28
     STA BAS2H
FC7E: A4 21
     LDY WNDWDTH  INIT Y TO RIGHTMOST INDEX
FC80: 88
     DEY  OF SCROLLING WINDOW
FC82: 69 0B
     ADC #$F0  INCR LINE NUMBER
FC84: C5 23
     CMP WNDBTM  DONE?
FC86: B0 0D
     BCS SCRL3  YES, FINISH
FC88: 48
     PHA
FC8A: 20 24
     FC JSR VTABZ  FORM BASL,H (BASE ADDR)
FC8C: B1 28
     SCRL2 LDA (BASL),Y  MOVE A CHR UP ON LINE
FC8E: 91 2A
     STA (BAS2L),Y
FC90: 88
     DEY  NEXT CHAR OF LINE
FC92: 10 F9
     BPL SCRL2
FC94: 30 E1
     BMI SCRL1  NEXT LINE (ALWAYS TAKEN)
FC96: A0 00
     SCRL3 LDY #$00  CLEAR BOTTOM LINE
FC98: 90 F0
     JSR CLREOL  GET BASE ADDR FOR BOTTOM LINE
FC9A: B0 86
     BCS VTAB  CARRY IS SET
FC9C: A4 24
     CLREOL LDY CH  CURSOR H INDEX
FC9E: A9 A0
     CLEOL2 LDA #$00  STORE BLANKS FROM 'HERE'
FC00: 91 28
     CLEOL2 STA (BASL),Y  STORE BLANKS FROM 'HERE'
FC02: C4 21
     CPY WNDWDTH
FC04: 9F F9
     BCC CLEOL2
FC06: 60
     RTS
FC08: 38
     WAIT SBC
FC0A: 48
     WAIT2 PHA
FC0C: E9 01
     WAIT3 SBC #$01
FC0E: D0 FC
     BNE WAIT3 1.0204 USEC
FC10: 68
     PLA
FC12: E9 01
     SBC #$01
FC14: D0 F6
     BNE WAIT2
FC16: 60
     RTS
FC18: E6 42
     NXTA4 INC A4L  INCR 2-BYTE A4
FC1A: D0 02
     BNE NXTA1  AND A1
FC1C: E6 43
     INC A4H
FC1E: A5 3C
     NXTA1 LDA A1L  INCR 2-BYTE A1.
FC20: C5 3E
     CMP A2L
FC22: A5 3D
     LDA A1H  AND COMPARE TO A2
84
FCC0: E5 3F  SBC  A2H  (CARRY SET IF &gt;=)
FCC2: E6 3C  INC  A1L
FCC4: D0 02  BNE  RTS4B
FCC6: E6 3D  INC  A1H
FCC8: 60  RTS
FCC9: A0 4B  HEADR  LDY  #$4B  WRITE A*256 'LONG 1'
FCCB: 20  DB  FC  JSR  ZERDLY  HALF CYCLES
FCCD: D0 09  BNE  HEADR  (650 USEC EACH)
FCD0: 69  PE  ADC  #$FE
FCD2: B0  F5  BCS  HEADR  THEN A 'SHORT 0'
FCD4: A0 21  LDY  #$21  OF 250 USEC ('0')
FCD6: 60  RTS
FCD9: A0 4B  HEADR  LDY  #$4B  WRITE A*256 'LONG 1'
FCDB: 20  DB  FC  JSR  ZERDLY  HALF CYCLES
FCD7: 85  2F  EOR  LASTIN
FCD9: 85  2F  STA  LASTIN  ECHO USER LINE
FCDB: 60  RTS
FCC: A2 08  RDBYTE  LDX  #$08  8 BITS TO READ
FCD6: 69  F9  ADC  #$FE
FCD8: B0  F5  BCS  HEADR  THEN A 'SHORT 0'
FCD1: 45 2F  EOR  LASTIN
FCD3: 10  F8  BPL  RDBIT
FCD5: 45 2F  EOR  LASTIN
FCD7: 85 2F  STA  LAST1  TIMING LOOP
FCD9: C8  INY  OR  500 USEC ('0')
FCDA: C8  INY  OR  500 USEC ('0')
FCD9: 88  ONELDY  DEY
FCD3: D0  FD  BNE  RDBYTE2
FCD6: 69  FE  ADC  #$FE
FCD8: B0  F5  BCS  HEADR
FCD9: 88 ONELDY  DEY
FCD2: 48  RDBYTE2  PHA  READ TWO TRANSITIONS
FCD6: 69  FE  ADC  #$FE
FCD8: B0  F5  BCS  HEADR
FCD9: 88 ONELDY  DEY
FCD2: 48  RDBYTE2  PHA  READ TWO TRANSITIONS
FCD6: 69  FE  ADC  #$FE
FCD8: B0  F5  BCS  HEADR
FCD9: 88 ONELDY  DEY
FCD2: 48  RDBYTE2  PHA  READ TWO TRANSITIONS
FCD6: 69  FE  ADC  #$FE
FCD8: B0  F5  BCS  HEADR
FCD9: 88 ONELDY  DEY
FCD2: 48  RDBYTE2  PHA  READ TWO TRANSITIONS
FCD6: 69  FE  ADC  #$FE
FCD8: B0  F5  BCS  HEADR
FCD9: 88 ONELDY  DEY
FCD2: 48  RDBYTE2  PHA  READ TWO TRANSITIONS
FCD6: 69  FE  ADC  #$FE
FCD8: B0  F5  BCS  HEADR
FCD9: 88 ONELDY  DEY
FCD2: 48  RDBYTE2  PHA  READ TWO TRANSITIONS
FCD6: 69  FE  ADC  #$FE
FCD8: B0  F5  BCS  HEADR
FCD9: 88 ONELDY  DEY
FCD2: 48  RDBYTE2  PHA  READ TWO TRANSITIONS
FCD6: 69  FE  ADC  #$FE
FCD8: B0  F5  BCS  HEADR
FCD9: 88 ONELDY  DEY
FCD2: 48  RDBYTE2  PHA  READ TWO TRANSITIONS
FCD6: 69  FE  ADC  #$FE
FCD8: B0  F5  BCS  HEADR
FCD9: 88 ONELDY  DEY
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 SETMODE 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 BPL   LT2
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
FE32:  90 F7 BCC   MOVE
FE34:  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:  B1 3C STA   (A4L),Y    (A4)
FE40:  20 DA FD JSR   PRBYTE
FE42:  A9 A0 LDA   #$0A
FE44:  20 ED FD JSR   NXTA4
FE46:  90 FF LDA   #$FF
FE48:  85 3E STY   INVFLG
FE4A:  60 RTS
FE4C:  20 75 FE LIST   TXA              IF USER SPEC'D ADR
FE4E:  A9 14 LDA   #$14         PC IF SPEC'D AND
FE50:  48 LIST2   PHA                DISEMBLE 20 INSTRS
FE52:  20 D0 FF JSR   INSTDSP
FE54:  20 EE FD JSR   PCADJ      ADJUST PC EACH INSTR
FE56:  84 30 STA   PCL
FE58:  84 3B STY   PCH
FE5A:  68 PLA
FE5C:  38 SEC
FE5E:  E9 01 SBC   #$01        NEXT OF 20 INSTRS
FE60:  D0 0F BNE   LIST2
FE62:  60 RTS
FE64:  8A A1PC    TXA              IF USER SPEC'D ADR
FE66:  F0 07 BEQ   A1PCRTS    COPY FROM A1 TO PC
FE68:  B5 3C A1PCLP   LDA   A1L,X
FE6A:  95 3A STA   PCL,X
FE6C:  CA DEX
FE6E:  10 F9 BPL   A1PCLP
FE70:  85 3E A1PCRTS   RTS
FE72:  60 RTS
FE74:  88 87 SRET   LDA   #$53F      SET FOR INVERSE VID
FE76:  88 5A STA   A2L
FE78:  E9 00 STX   LDX   #$53F      SET FOR INVERSE VID
FE7A:  A0 1F LDA   #$00        SIMULATE PORT #0 INPUT
FE7C:  85 3E IMPORT STA   A2L    SPECIFIED (KEYIN ROUTINE)
FE7E:  A2 38 INPRT   LDX   #$KSNL
FE80:  A2 38 LDA   #$KSNL
FE82:  A0 1B LDY   #$KEYIN
FE84:  D0 08 BNE   IORPT
FE86:  A9 00 LDA   #$500      SIMULATE PORT #0 OUTPUT
FE88:  85 3E OUTPORT STA   A2L    SPECIFIED (COUT1 ROUTINE)
FE8A:  A2 38 OUTPT   LDX   #$CSCW
FE8C:  A0 F0 LDA   #$COUT1
FE8E:  A5 3E IOPRT   LDA   A2L    SFT RAM IN/OUT VECTORS
FE90:  29 0F AND   #$50F
FE92:  F0 06 BNE   IOPRT1
FE94:  09 C0 LDA   #$10ADR/256
FE96:  A0 00 LDA   #$500
FE98:  F0 02 BNE   IOPRT2
FE9A:  A9 FD LDA   #$COUT1/256
FE9C:  94 00 IOPRT2   STY   LOC0,X
FE9E:  9D 01 STA   LOC1,X
FEA0:  60 RTS
FEA2:  EA NOP
FEA4:  EA NOP
FEA6:  4C 00 E0 XBASE   JMP   BASIC        TO BASIC WITH SCRATCH
FEA8:  4C 03 E0 BASECONT   JMP   BASIC2 CONTINUE BASIC
FEB0:  87

87
FEB6: 20 75 FE GO JSR A1PC ADR TO PC IF SPEC'D
FEB9: 20 3F FF JSR RESTORE RESTORE META REGS
FEBC: 4C D7 FA REGZ JMP REGDSP TO REG DISPLAY
FEC2: C6 34 TRACE DEC YSAV
FEC4: 20 75 FE STEPZ JSR A1PC ADR TO PC IF SPEC'D
FEC7: 4C 43 FA JMP STEP TAKE ONE STEP
FEC9: 4C F8 03 USR JMP USRADR TO USR USER AT USRADR
FECD: A9 40 WRITE LDA #$40
FECF: 20 C9 FC JSR HEADR WRITE 10-SEC HEADER
FED2: A0 27 LDY #$27
FED4: A2 00 LDX LDX #$00
FED6: A1 3C EOR (A1L,X)
FED8: F0 4D BEQ BELL
FED9: A0 1D LDY #$1D
FEB1: 4C D7 FA REGZ JMP REGDSP TO REG DISPLAY
FED3: A0 3B LDY #$3B INDEX FOR 0/1 TEST
FED5: 20 EC FC JSR RDBYTE READ A BYTE
FED7: 45 2E EOR CHKSUM
FED9: 85 2E STA CHKSUM UPDATE RUNNING CHKSUM
FEDE: 20 BA FC JSR WRBYTE WRITE 10-SEC HEADER
FEB0: A0 22 LDY #$22
FEB2: F0 4D BEQ BELL
FED4: 20 C9 FC JSR HEADR WRITE 10-SEC HEADER
FED6: A1 3C EOR (A1L,X)
FED8: F0 4D BEQ BELL
FEB1: 4C D7 FA REGZ JMP REGDSP TO REG DISPLAY
FED3: A0 3B LDY #$3B INDEX FOR 0/1 TEST
FED5: 20 EC FC JSR RDBYTE READ A BYTE
FED7: 45 2E EOR CHKSUM
FED9: 85 2E STA CHKSUM UPDATE RUNNING CHKSUM
FEDE: 20 BA FC JSR WRBYTE WRITE 10-SEC HEADER
FEB0: A0 22 LDY #$22
FEB2: F0 4D BEQ BELL
FEB4: 20 C9 FC JSR HEADR WRITE 10-SEC HEADER
FEB6: A1 3C EOR (A1L,X)
FED8: 88
FF70: 20 C7 FF JSR ZMODE CLEAR MON MODE, SCAN IDX
FF73: 20 A7 FF NXTITM JSR GETNUM GET ITEM, NON-HEX
FF76: 84 34 STY YSAV CHAR IN A-REG
FF78: A0 17 LDY #$17 X-REG=0 IF NO HEX INPUT
FF7A: 88 CHRSRCH DEY
FF7B: 30 E8 BMI MON NOT FOUND, GO TO MON
FF7D: D9 CC FF CMP CHRTBL,Y FIND CMND CHAR IN TEL
FF80: D0 F8 BNE CHRSRCH
FF82: 20 BE FF JSR TOSUB FOUND, CALL CORRESPONDING
FF85: A4 34 LDY YSAV SUBROUTINE
FF87: 4C 73 FF JMP NXTITM
FF8A: A2 03 DIG LDX #$03
FF8C: 0A ASL A
FF8E: 0A ASL A GOT HEX DIG,
FF90: 0A ASL A SHIFT INTO A2
FF92: 0A ASL A
FF94: 0A ASL A
FF96: 0A ASL A
FF98: 0A ASL A
FF9A: 0A ASL A
FF9C: 0A ASL A
FF9E: 0A ASL A
FFA0: 0A ASL A
FFA2: BE 3F FF JMP NXTBAS
FFA4: A5 31 NXTBAS LDA MODE
FFA6: D0 06 BNE NXTBS2 IF MODE IS ZERO
FFA8: 86 3F STX A2H, X THEN COPY A2 TO A3
FFAA: 85 31 LDA A2H,X
FFAC: 85 41 STA A3H,X
FFAD: E8 INX
FFAE: A9 F3 FE TOSUB LDA #GO/256 PUSH HIGH-ORDER
FFEC: 17 DFB SETMODE-1
FFED: 17 DFB SETMODE-1
FFEE: 2B DFB MOVE-1
FFFFFF: 1F DFB LT-1
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
FFFB: 03 DFB NMI/256
FFFC: 59 DFB RESET RESET VECTOR
FFFF: FF DFB RESET/256
FFFF: 86 DFB IRQ IRQ VECTOR
FFFF: FA DFB IRQ/256
XQTNZ EQU $3C
***************
*             *
*  APPLE-II  *
*  MINI-ASSEMBLER *
*             *
* COPYRIGHT 1977 BY *
* APPLE COMPUTER INC. *
*             *
* ALL RIGHTS RESERVED *
*             *
* S. WOZNIAK *
* A. BAUM *
***************

TITLE "APPLE-II MINI-ASSEMBLER"
FORMAT EQU $2E
LENGTH EQU $2F
MODE EQU $31
PROMPT EQU $33
YSAV EQU $34
L EQU $35
PCL EQU $3A
PCH EQU $3B
A1H EQU $3D
A2L EQU $3E
A2H EQU $3F
A4L EQU $42
A4H EQU $43
FMT EQU $44
IN EQU $200
INSDS2 EQU $F88E
INSTDSP EQU $F8D0
PRBL2 EQU $F94A
PCADJ EQU $F953
CHAR1 EQU $F9B4
CHAR2 EQU $FA00
MNEML EQU $F9C0
MNEMR EQU $FA00
CURSUP EQU $FC1A
GETLNZ EQU $FD67
COUT EQU $FDED
BL1 EQU $FE00
A1PCLP EQU $FE78
BELL EQU $FF3A
GETNUM EQU $FFA7
ZMODE EQU $FFC7
CHRTBL EQU $FFCC
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: 1E 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

91
F517:  E5 3B              SBC   PCH
F519:  D0 6B              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              STA   (PCL),Y
F522:  88                DEY
F523:  10 F8              BPL   FNDOP2
F525:  20 1A FC           JSR   CURSUP
F528:  20 1A FC           JSR   CURSUP   RESTORE CURSOR
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:  A0 17              LDY   #$17   INIT DELIMITER INDEX
F544:  30 4B              BMI   RESETZ     ERR IF UNRECOGNIZED DELIM
F547:  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   MODE
F550:  A0 00              LDY   #$0
F552:  8A                 TXA
F554:  AA                 TAX
F556:  20 0E FF  FAKEMON JSR   GETNUM READ PARAM
F559:  4C 95 F5           JMP   NXTLINE    GET NEXT LINE
F55C:  A5 3D     TRYNEXT LDA A1H GET TRIAL OPCODE
F55E:  20 8E F8           JSR   INSDS2     GET FMT+LENGTH FOR OPCODE
F561:  AA                 TAX
F562:  BD 00 FA LDA MNEMR,X GET LOWER MNEMONIC BYTE
F565:  C5 42              CMP   A4L MATCH?
F567:  D0 13              BNE   TRYNEXT
F569:  BD C0 F9           LDA   MNEML,X GET UPPER MNEMONIC BYTE
F56C:  C5 43              CMP   A4H MATCH?
F56E:  D0 0C              BNE   TRYNEXT
F570:  A5 44              LDA   FMT GET TRIAL FORMAT
F572:  A4 2E              LDY   FORMAT GET TRIAL FORMAT
F574:  C0 9D              CPY   #$9D TRIAL FORMAT RELATIVE?
F576:  F0 88              BEQ   REL YES.
F578:  C5 2E     NREL    CMP   FORMAT SAME FORMAT?
F57A:  F0 9F              BEQ   FINDOP YES.
F57C:  C6 3D     NEXTOP DEC   A1H NO, TRY NEXT OPCODE
F57E:  D0 DC              BNE   TRYNEXT
F580:  E6 44              INC   PMT NO MORE, TRY WITH LEN=2
F582:  C6 35              DEC   L WAS L=2 ALREADY?
F584:  F0 D6              BEQ   TRYNEXT NO.
F586:  A4 34     ERR    LDA   YSAV   YES, UNRECOGNIZED INST.
F588:  98     ERR2    STA   ERR2
F589:  AA                 TAX
F58A:  20 4A P9  FAKEMON JSR   PUBL2 PRINT ^ UNDER LAST READ
F58D:  A9 DE              LDA   #$DE CHAR TO INDICATE ERROR
F58F:  20 ED FD           LSR   COUT POSITION.
F592:  20 3A FF  RESETZ JSR   BELL '!' INIT SCREEN PROMPT
F595:  85 33              STA   PROMPT INITIALIZE PROMPT
F599:  20 67 FD           LSR   GETLNZ GET LINE.
F59C:  20 77 FF           LSR   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 '?' IN COL 1?
F5A9:  F0 92              BEQ   PAKEMON YES, SIMULATE MONITOR
F5AB:  88                DEY NO, BACKUP A CHAR
F5AC:  20 A7 FF  FAKEMON JSR   GETNUM GET A NUMBER
F5AF:  C9 93              CMP   #$93 :' TERMINATOR?
F5B1:  D0 D5              BNE   ERR4 ERR2 NO, ERR.
F5B3:  8A                 TXA
F5B4:  F0 D2              BEQ   ERR2 NO ADR PRECEDING COLON.
F5B6:  20 7B FR           LSR   A1CFLP MOVE ADR TO PCL, PCH.
F5B8:  A9 03              SPACE LDA #$3 COUNT OF CHAR IN MNEMONIC
F5BB:  85 3D              STA   A1H
F5BD:  20 34 P6  NXTMN JSR   GETNSP GET FIRST MNEM CHAR.
F5C0:  9A                 NXTM A
F5C1:  E9 BE              SBC   #$BE SUBTRACT OFFSET
F5C3:  C9 C2              CMP   #$C2 LEGAL CHAR?
F5C5:  90 C1              BCC   ERR2 NO.
F5C7:  0A A9              ASSL A COMpress-LEFT JUSTIFY
F5C8:  0A A9              ASSL A
F5C9:  A2 04              LDX   #$4
F5CB:  0A                 NXTM A DO 5 TRIPLE WORD SHIFTS
F5CC: 26 42              ROL   A4L
F5CE: 26 43              ROL   A4H
F5D0: CA                 DEX
F5D1: 10 F8              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 F6 FORM2    JSR   GETNSP     GET FIRST CHAR OF ADDR
F5DE: 84 34 STY   YSAV
F5E0: CA                 DEX
F5E3: D0 13              BNE   FORM3      NO
F5E5: 20 34 F6 JSR   GETNSP    YES, GET SECOND CHAR
F5E8: DD BA F9           CMP   CHAR2,X     MATCHES SECOND HALF?
F5EB: F0 0D              BEQ   FORM5      YES.
F5E7: E4                 INX    NO, IS SECOND HALF ZERO?
F5E9: A2 05 FORM1       LDX   #$5      5 CHARs IN ADDR MODE
F5EB: 20 34 F6 JSR   GETNSP    YES.
F5F0: F0 07               BEQ   FORM4      YES.
F5F2: C9 A4              CMP   #$A4       NO, SECOND HALF OPTIONAL?
F5F5: F0 03              BEQ   FORM4      YES.
F5F8: 18 FORM3       CLC     CLEAR BIT-NO MATCH
F5F9: 88 FORM4       DEY     BACK UP 1 CHAR
F5FA: 26 44 FORM5       ROL   FORM    FORM FORMAT BYTE
F5FC: E0 03              CFX    #$3      TIME TO CHECK FOR ADDR.
F5FD: D0 0D              BNE   FORM7      NO
F5F0: 20 A7 FF JSR   GETNUM    YES
F5F3: A5 3F              LDA   A2H
F5F5: F0 01              BEQ   FORM6      HIGH-ORDER BYTE ZERO
F5F7: E8                 INX    NO, INCR FOR 2-BYTE
F5F8: 86 35 FORM6       STX   L       STORE LENGTH
F5FB: A2 03              LDX   #$3      RELOAD FORMAT INDEX
F5FC: 88                 DEY    BACKUP A CHAR
F5FD: 86 3D FORM7       STX   A1H      SAVE INDEX
F5FE: CA                 DEX    DONE WITH FORMAT CHECK?
F600: 10 C9              BPL   FORM2      NO.
F602: A5 44              LDA   FMT      YES, PUT LENGTH
F604: 0A                 ASL   A       IN LOW BITS
F606: 0A                 ASL   A
F608: 05 35              ORA   L
F60A: C9 20              CMP   #$20      ADD "$" IF NONZERO LENGTH
F60C: B0 06              BCS   FORM8     ADD "$" IF NONZERO LENGTH
F60E: A0                 ASL   A       AND DON'T ALREADY HAVE IT
F610: 05 35              ORA   L
F612: C9 20              CMP   #$20      ADD "$" IF NONZERO LENGTH
F614: A0                 ASL   A       AND DON'T ALREADY HAVE IT
F616: 05 35              ORA   L
F618: C9 20              CMP   #$20      ADD "$" IF NONZERO LENGTH
F61A: B0 06              BCS   FORM8     ADD "$" IF NONZERO LENGTH
F61C: A6 35              LDX   L       AND DON'T ALREADY HAVE IT
F61E: F0 02              BEQ   FORM8
F620: 09 80              ORA   #$80
F622: 85 44 FORM8       STA   FMT
F624: 84 34              STY   YSAV
F626: B9 00 02             LDA   IN,Y     GET NEXT NONBLANK
F628: C9 BB              CMP   #$BB     "' START OF COMMENT?
F62A: F0 04              BEQ   FORM9      YES
F62C: C9 8D              CMP   #$8D     CARRIAGE RETURN?
F62E: D0 80              BNE   ERR4      NO, ERR.
F630: 4C 5C F5 FORM9    JMP   TRYNEXT
F632: 09 80              ORA   #$80
F634: B9 00 02             LDA   IN,Y     GET NEXT NONBLANK
F636: 84 34              STY   YSAV
F638: C9 A0              CMP   #$A0     GET NEXT NON BLANK CHAR
F63A: F0 08              BEQ   GETNSP
F63C: 60                 RTS
F63E: ORG   $F666
F666: 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 ADD1 DEX INDEX TO NEXT MORE SIGNIF. BYTE.
F431: 60 RTS RETURN
F432: 06 F3 MD1 ASL SIGN CLEAR LSB OF SIGN.
F434: 20 37 F4 JSR ABSSWP ABS VAL OF M1, THEN SWAP WITH M2
F437: 24 F9 ABSSWP BIT M1 MANT1 NEGATIVE?
F439: 10 05 BPL ABSWAP1 NO, SWAP WITH MANT2 AND RETURN.
F43B: 20 A4 F4 JSR FCMPUL YES, COMPLEMENT IT.
F43E: B6 F3 INC SIGN INCR SIGN, COMPLEMENTING LSB.
F440: 38 ABSSWAP1 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 #$8E INIT EXP1 TO 14,
F453: 85 F8 STA X1 THEN NORMALIZE TO FLOAT.
F455: A5 F9 NRM1 LDA M1 HIGH-ORDER MANT1 BYTE.
F457: C9 C0 CMP #$C0 UPPER TWO BITS UNEQUAL?
F459: 30 0C BMI RTS1 YES, RETURN WITH MANT1 NORMALIZED
F45B: C6 F8 DBC X1 DECUMENT EXP1.
F45D: 06 F8 ASL M1+2
F45F: 26 FA ROL M1+1 SHIFT MANT1 (3 BYTES) LEFT.
F461: 26 F9 ROL M1
F463: A5 F8 NRM LDA X1 EXP1 ZERO?
F465: D0 BE BNE NRM1 NO, CONTINUE NORMALIZING.
F467: 60 RTS RTS1 RETURN
F468: 20 A4 F4 FSUB JSR FCMPUL CMPL MANT1,CLEARS CARRY UNLESS 0
F46B: 20 7B F4 SWPALGN JSR ALGNMSP 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: AS F9 RTAR LDA M1 SIGN OF MANT1 INTO CARRY FOR
F47D: 0A ASL RIGHT ARITH SHIFT.
F480: E6 F8 RTLOG INC X1 INCR X1 TO ADJUST FOR RIGHT SHIFT
F482: F0 75 BBQ OVFL EXP1 OUT OF RANGE.
F484: A2 FA RTLOG1 LDX $5FA INDEX FOR 6:BYTE RIGHT SHIFT.
F486: 76 FF ROR1 ROR E+3,X
F488: E6 F8 RTLOG INC X1 TO ADJUST FOR RIGHT SHIFT.
F48A: 20 25 F4 JSR ROR1 LOOP UNTIL DONE.
F48B: 60 RTS RETURN.
F48C: 20 32 F4 FMUL JSR MD1 ABS VAL OF MANT1, MANT2
F48F: 65 F8 ADC X1 ADD EXP1 TO EXP2 FOR PRODUCT EXP
F491: 20 E2 F4 JSR ADD ADD MULTICAND TO PRODUCT.
F494: 18 CLC CLEAR CARRY FOR FIRST BIT.
F496: F0 75 BEQ OVFL EXP1 OUT OF RANGE.
F498: A2 FA RTLOG1 LDX #$FA INDEX FOR 6:BYTE RIGHT SHIFT.
F49A: 20 25 F4 JSR ROR1 LOOP UNTIL DONE.
F49B: 60 RTS RETURN.
F49C: 20 32 F4 PMUL JSR MD1 ABS VAL OF MANT1, MANT2
F49F: 65 F8 ADC X1 ADD EXP1 TO EXP2 FOR PRODUCT EXP
F4A1: 20 E2 F4 JSR ADD ADD MULTICAND TO PRODUCT.
F4A4: 38 FCOMPL SBC SET CARRY FOR SUBTRACT.
F4A5: A2 03 LDX #$3 INDEX FOR 3 BYTE SUBTRACT.
F4A7: A9 00 COMPL1 LDA #$0 CLEAR A.
F4A9: F5 F8 SBC X1,X SUBTRACT BYTE OF EXP1.
F4AB: 95 F8 STA X1,X RESTORE IT.
F4AD: CA DEX NEXT MORE SIGNIFICANT BYTE.
F4AE: DO F7 BNE COMPL1 LOOP UNTIL DONK.
F4BF: F0 C5 BBQ ADDEND NORMALIZE (OR SHIPT BT IF OVFL).
F4C0: 20 32 F4 FDIV JSR MD1 TAKE ABS VAL OF MANT1, MANT2.
F4C2: 20 E2 F4 JSR ROR1 LOOP UNTIL DONE.
F4C4: 18 CLC CLEAR CARRY FOR FIRST BIT.
F4C5: F0 75 BEQ OVFL EXP1 OUT OF RANGE.
F4C7: 68 DIV3 PLA FULL BYTE OF DIFFERENCE OFF STACK
F4C8: 90 02 BCC DIV4 IF M2<0 THEN DON'T RESTORE M2.
F4CA: 95 FF STA M2+3,X
F4CC: E8 DIV4 INX NEXT LESS SIGNIFICANT BYTE.
F4CD: D0 F8 BNE DIV3 LOOP UNTIL DONK.
F4CF: 26 F8 ROL M1+2
F4D0: 26 FA ROL M1+1 ROLL QUOTIENT LEFT, CARRY INTO LSB
F4D3: 26 F9 ROL M1
F4D5: 06 FF ASL M2+2
F4D7: 26 F6 ROL M2+1 SHIFT DIVIDEND LEFT
F4D9: 26 F5 ROL M2
F4DA: B0 1C BCS OVFL OVFL IS DUE TO UNNORMED DIVISOR
F4DB: 88 DEY NEXT DIVIDE ITERATION.
F4DD: DO DA BNE DIV1 LOOP UNTIL DONK 23 ITERATIONS.
F4E0: F0 BE BBQ MEND NORM. QUOTIENT AND CORRECT SIGN.
F4E2: 86 F8 MD2 STX M1+2
F4E4: 86 FA STX M1+1 CLEAR MANT1 (3 BYTES) FOR MUL/DIV.
F4E6: 86 F9 STX M1
F4E8: B0 0D BCS OVCHK IF CALC SET CARRY, CHECK FOR OVFL
F4EA: 30 04 BMI MD3 IF NEG THEN NO UNDERFLOW.
F4EC: 68 PLA POP ONE RETURN LEVEL.
F4ED: 68 PLA
F4EE: 90 B2 BCC NORMX CLEAR X1 AND RETURN.
F4F0: 49 80 MD3 EOR $800 COMPLEMENT SIGN BIT OF EXPONENT.
F4F2: 85 F8 STA X1 STORE IT.
F4F4: A0 17 LDY $17 COUNT 24 MUL/23 DIV ITERATIONS.
F4F6: 60 RTS RETURN.
F4F7: 10 F7 OVCHK BPL MD3 IF POSITIVE EXP THEN NO OVFL.
F4F9: 4C F5 03 OVFL JMP OVLOC ORG #$F63D
F4F9: 4C F5 03 OVFL JMP OVLOC ORG #$F63D
F503: 26 7D F4 FIX1 JSR RTAR
F506: A5 F8 FIX LDA X1
F509: 24 F9 BIT M1
F514: 10 0A BPL FIXRTS
F516: 80 00 FIXRTS RTS
F518: A9 00 UNDFL LDA $0
F51A: 85 F9 STA M1
F51C: 85 FA STA M1+1
F51E: 60 RTS
TITLE "SWEET16 INTERPRETER"

R0L EQU $0
R0H EQU $1
R14H EQU $1D
R15L EQU $1E
R15H EQU $1F
SW16PAG EQU $F7
SAVE EQU $FF4A
RESTORE EQU $FF3F
ORG $F689

F689: 20 4A FF SW16 JSR SAVE PRESERVE 6502 REG CONTENTS
F68D: 85 1E STA R15L INIT SWEET16 PC
F68F: 68 PLA FROM RETURN
F690: 85 1F STA R15H ADDRESS
F692: 20 98 F6 SW16B JSR SW16C INTERPRET AND EXECUTE
F694: 4C 92 F6 JMP SW16B ONE SWEET16 INSTR.
F696: E6 1E SW16C INC R15L
F698: D0 02 BNE SW16D INCR SWEET16 PC FOR FETCH
F69A: E6 1F INC R15H
F69C: A9 F7 SW16D LDA #SW16PAG
F69E: 48 PHA PUSH ON STACK FOR RTS
F6A0: 20 4A FF SW16 JSR SAVE PRESERVE 6502 REG CONTENTS
F6A4: B1 1E LDA (R15L),Y FETCH INSTR
F6A5: 29 0F AND #$F MASK REG SPECIFICATION
F6A7: 0A ASL A DOUBLE FOR TWO BYTE REGISTERS
F6A8: AA TAX TO X REG FOR INDEXING
F6A9: 4A LSR A
F6AA: 51 1E EOR (R15L),Y NOW HAVE OPCODE
F6AC: F0 0B BSR TOBR IF ZERO THEN NON-REG OP
F6AE: 86 1D STX R14H INDICATE 'PRIOR RESULT REG'
F6B0: 4A LSR A
F6B1: 4A LSR A OPCODE*2 TO LSB'S
F6B2: 4A LSR A
F6B3: A8 TAY TO Y REG FOR INDEXING
F6B4: B9 E1 F6 LDA OPTBL-2,Y LOW ORDER ADR BYTE
F6B7: 48 PHA ONTO STACK
F6B8: 60 RTS GOTO REG-OP ROUTINE
F6BA: E6 1E TOBR INC R15L
F6BB: D0 02 BNE TOBR2 INCR PC
F6BD: E6 1F INC R15H
F6BF: BD E4 F6 TOBR2 LDA BRTBL,X LOW ORDER ADR BYTE
F6C2: 48 PHA ONTO STACK FOR NON-REG OP
F6C3: A5 1D LDA R14H 'PRIOR RESULT REG' INDEX
F6C5: 4A LSR A PREPARE CARRY FOR BC, BNC.
F6C6: 60 RTS GOTO NON-REG OP ROUTINE
F6C7: 68 RTNZ PLA POP RETURN ADDRESS
F6C8: 68 PLA
F6C9: 20 3F FF JSR RESTORE RESTORE 6502 REG CONTENTS
F6CC: 6C 1E 00 JMP (R15L) RETURN TO 6502 CODE VIA PC
F6CF: B1 1E SETZ LDA (R15L),Y HIGH-ORDER BYTE OF CONSTANT
F6D1: 95 01 STA R0H,X
F6D3: 88 DEY
F6D4: B1 1E LDA (R15L),Y LOW-ORDER BYTE OF CONSTANT
F6D6: 95 00 STA ROL,X
F6D8: 98 THA Y-REG CONTAINS 1
F6DA: 65 1E ADC R15L ADD 2 TO PC
F6DC: 85 01 STA R15L
F6DE: 90 02 BCC SET2
F6E0: E6 1F INC R15H
F6E2: 60 SET2 RTS
F6E3: 02 OPTBL DBF SET-1 1X
F6E4: F9 BKTBEL DBF RTN-1 0
F6E5: 04 DBF LD-1 2X
F6E6: 9D DBF BR-1 1
F6E7: 0D DBF ST-1 3X
F6E8: 9E DBF BNC-1 2
F6E9: 25 DBF LDAT-1 4X
F6EA: AP DBF RC-1 3
F6EB: 16 DBF STAT-1 5X
F6EC: B2 DBF BP-1 4
F6ED: 47 DBF LDDAT-1 6X
F6EE: B9 DBF BM-1 5
F6EF: 51 DBF STDAT-1 7X
F6F0: C0 DBF BZ-1 6
F6F1: 2F DBF POP-1 8X
F6F2: C9 DBF BNZ-1 7
F6F3: 5B DBF STPAT-1 9X
F6F4: D2 DBF BMI-1 8
F6F5: 85 DBF ADD-1 AX
F6F6: DD DBF BNM1-1 9
F6F7: 6E DBF SUB-1 BX
F6F8: 05 DBF BK-1 A
F6F9: 33 DBF POPD-1 CX
F6FA: E8 DBF RS-1 B
F6FB: 70 DBF CPR-1 DX
F6FC: 93 DBF BS-1 C
F6FD: 1E DBF INR-1 EX
F6FE: 9E DBF BNC-1 D
F6FF: 25 DBF BR-1 FX
F700: E6 1F INC R15H
F701: 85 DBF BR-1 BX
F702: 33 DBF POPL-1 CX
F703: 10 CA SET BPL SETZ ALWAYS TAKEN
F705: B5 00 LD ROL,X sta rol,x
F706: 00 BK BQU *-1
F707: 85 00 STA ROL
F709: B5 01 LDA ROL,X MOVE RX TO RO
F70B: 85 01 STA ROL
F70D: 60 RTS
F70E: A5 00 ST LDA ROL
F710: 95 00 STA ROL,X MOVE RO TO RX
F712: A5 01 LDA ROL
F714: 95 01 STA ROL,X
F716: 60 RTS
F717: A5 00 STAT LDA ROL
F719: 81 00 STAT2 STA ROL,X STORE BYTE INDIRECT
F71B: A0 00 LOY #$0
F71D: 84 1D STAT3 STY R14H INDICATE R0 IS RESULT NEG
F71F: 06 00 INR INC ROL,X
F721: D0 02 INR2 INC RX
F723: F0 06 BEQ STAT3 ALWAYS TAKEN
F725: 60 INR2 RTS
F726: A1 00 LDAT LDA (ROL,X) LOAD INDIRECT (RX)
F728: 85 00 STA ROL TO RO
F72A: A0 00 LOY #$0
F72C: 84 01 STY ROH ZERO HIGH-ORDER RO BYTE
F72E: 05 0E BEQ STAT3 ALWAYS TAKEN
F730: A0 00 POP LDY #$0 HIGH ORDER BYTE = 0
F732: 00 0E BEQ POP2 ALWAYS TAKEN
F734: 20 66 07 POPD JSR DCR DCR RX
F737: A1 00 LDA (ROL,X) POP HIGH ORDER BYTE &RX
F739: A8 TAY SAVE IN Y-REG
F73A: 20 66 07 POP2 JSR DCR DCR RX
F73D: A1 00 LDA (ROL,X) LOW-ORDER BYTE
F73F: 85 00 STA ROL TO RO
F741: 84 01 STY ROH
F743: A0 00 POP3 LDY #$0 INDICATE RO AS LAST RESULT REG
F745: 84 1D STY R14H
F747: 60 RTS
F748: 20 26 07 LDDAT JSR LDAT LOW-ORDER BYTE TO RO, INCR RX
F74B: A1 00 LDA (ROL,X) HIGH-ORDER BYTE TO RO
F74D: 85 01 STA ROH
F74F: 0C 1F 07 JMP INR INCR RX
F752: 20 17 07 STDAT JSR STAT STORE INDIRECT LOW-ORDER
F7EB: 20 66 F7           JSR   DCR        DCR STACK POINTER
F7EE: A1 00              LDA   (R0L,X)    POP HIGH RETURN ADDRESS TO PC
F7F0: 85 1F              STA   R15H
F7F2: 20 66 F7           JSR   DCR        SAME FOR LOW-ORDER BYTE
F7F5: A1 00              LDA   (R0L,X)
F7F7: 85 1E              STA   R15L
F7F9: 60                 RTS
F7FA: 4C C7 F6 RTN      JMP   RTNZ
6502 MICROPROCESSOR INSTRUCTIONS

AOC  Add Memory to Accumulator with Carry
AND  "AND" Memory with Accumulator
ASL  Shift Left One Bit (Memory or Accumulator)
BCC  Branch on Carry Clear
BCS  Branch on Carry Set
BED  Branch on Result Zero
BIT  Test Bits in Memory with Accumulator
BMI  Branch on Result Minus
ONE  Branch on Result not Zero
BPL  Branch on Result Plus
BRK  Force Break
BVC  Branch on Overflow Clear
BVS  Branch on Overflow Set
CLC  Clear Carry Flag
CLD  Clear Decimal Mode
CLI  Clear Interrupt Disable Bit
CLV  Clear Overflow Flag
CMP  Compare Memory and Accumulator
CPX  Compare Memory and Index X
CPY  Compare Memory and Index Y
DEC  Decrement Memory by One
DEX  Decrement Index X by One
DEY  Decrement Index Y by One
FOR  "Exclusive-Or" Memory with Accumulator
INC  Increment Memory by One
INX  Increment Index X by One
INY  Increment Index Y by One
JMP  Jump to New Location
JSA  Jump to New Location Saving Return Address

LDA  Load Accumulator with Memory
LDX  Load Index X with Memory
LDY  Load Index Y with Memory
LSR  Shift Right one Bit (Memory or Accumulator)
NOP  No Operation
ORA  OR Memory with Accumulator
PHA  Push Accumulator on Stack
PHP  Push Processor Status on Stack
PLA  Pull Accumulator from Stack
PLP  Pull Processor Status from Slack
ROL  Rotate One Bit Left (Memory or Accumulator)
ROR  Rotate One Bit Right (Memory or Accumulator)
RTI  Return from Interrupt
RTS  Return from Subroutine
SBC  Subtract Memory from Accumulator with Borrow
SEC  Set Carry Flag
SED  Set Decimal Mode
SEI  Set Interrupt Disable Status
STA  Store Accumulator in Memory
STX  Store Index X in Memory
STY  Store Index Y in Memory
TAX  Transfer Accumulator to Index X
TAY  Transfer Accumulator to Index Y
TSX  Transfer Stack Pointer to Index X
TXA  Transfer Index X to Accumulator
TXS  Transfer Index X to Stack Pointer
TYA  Transfer Index Y to Accumulator
THE FOLLOWING NOTATION APPLIES TO THIS SUMMARY:

A  Accumulator
X, Y  Index Registers
M  Memory
C  Borrow
P  Processor Status Register
S  Stack Pointer
✓  Change
−  No Change
+  Add
∧  Logical AND
∩  Subtract
$  Logical Exclusive OR
↓  Transfer From Stack
←  Transfer To Stack
→  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 then 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 Code</th>
<th>No. Bytes</th>
<th>'P' Status Reg. N Z C I B V</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ADC</strong> Add memory to accumulator with carry</td>
<td>A - M - C -&gt; A C</td>
<td>Immediate</td>
<td>ADC #Op</td>
<td>69</td>
<td>2</td>
<td>√√√---√</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>ADC Op</td>
<td>62</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>ADC Op</td>
<td>75</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>ADC Op</td>
<td>60</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute, Y</td>
<td>ADC Op</td>
<td>78</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect,X)</td>
<td>ADC Op</td>
<td>62</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect,Y)</td>
<td>ADC Op</td>
<td>71</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>AND</strong> memory with accumulator</td>
<td>A &amp; M -&gt; A</td>
<td>Immediate</td>
<td>AND #Op</td>
<td>26</td>
<td>2</td>
<td>√√---√</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>AND Op</td>
<td>25</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>AND Op</td>
<td>35</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>AND Op</td>
<td>20</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute, Y</td>
<td>AND Op</td>
<td>30</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect,X)</td>
<td>AND Op</td>
<td>21</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect,Y)</td>
<td>AND Op</td>
<td>31</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>ASL</strong> Shift left one bit (memory or accumulator)</td>
<td>(See Figure 1)</td>
<td>Accumulator</td>
<td>ASL A</td>
<td>0A</td>
<td>1</td>
<td>√√√---√</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>ASL Op</td>
<td>0E</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>ASL Op</td>
<td>16</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>ASL Op</td>
<td>06</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute, Y</td>
<td>ASL Op</td>
<td>03</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect,X)</td>
<td>ASL Op</td>
<td>1E</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td><strong>BCC</strong> Branch on carry clear</td>
<td>Branch on C=0</td>
<td>Relative</td>
<td>BCC Op</td>
<td>90</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>BCS</strong> Branch on carry set</td>
<td>Branch on C=1</td>
<td>Relative</td>
<td>BCS Op</td>
<td>80</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>BEQ</strong> Branch on result zero</td>
<td>Branch on Z=1</td>
<td>Relative</td>
<td>BEQ Op</td>
<td>60</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BIT</strong> Test bits in memory with accumulator</td>
<td>A &amp; M, M3 -&gt; N, M0 -&gt; Y</td>
<td>Zero Page</td>
<td>BIT #Op</td>
<td>24</td>
<td>2</td>
<td>M3/Y---M0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>BIT Op</td>
<td>25</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>BIT Op</td>
<td>22</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td><strong>BMI</strong> Branch on result minus</td>
<td>Branch on N=1</td>
<td>Relative</td>
<td>BMI Op</td>
<td>30</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>BNE</strong> Branch on result not zero</td>
<td>Branch on Z=0</td>
<td>Relative</td>
<td>BNE Op</td>
<td>00</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>BPL</strong> Branch on result plus</td>
<td>Branch on N=0</td>
<td>Relative</td>
<td>BPL Op</td>
<td>10</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>BRK</strong> Force Break</td>
<td>Forced Interrupt</td>
<td>Implied</td>
<td>BRK #</td>
<td>00</td>
<td>1</td>
<td>--- ---</td>
</tr>
<tr>
<td><strong>BVC</strong> Branch on overflow clear</td>
<td>Branch on V=0</td>
<td>Relative</td>
<td>BVC Op</td>
<td>50</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

### Additional Instructions

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX Code</th>
<th>No. Bytes</th>
<th>'P' Status Reg. N Z C I B V</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BVS</strong> Branch on overflow set</td>
<td>Branch on V=1</td>
<td>Relative</td>
<td>BVS Op</td>
<td>70</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>CLC</strong> Clear carry flag</td>
<td>0 -&gt; C</td>
<td>Implied</td>
<td>CLC</td>
<td>10</td>
<td>1</td>
<td>--- ---</td>
</tr>
<tr>
<td><strong>CLD</strong> Clear decimal mode</td>
<td>0 -&gt; D</td>
<td>Implied</td>
<td>CLD</td>
<td>08</td>
<td>1</td>
<td>--- ---</td>
</tr>
<tr>
<td><strong>CLI</strong> Clear interrupt</td>
<td>0 -&gt; I</td>
<td>Implied</td>
<td>CLI</td>
<td>58</td>
<td>1</td>
<td>--- ---</td>
</tr>
<tr>
<td><strong>CLV</strong> Clear overflow flag</td>
<td>0 -&gt; Y</td>
<td>Implied</td>
<td>CLV</td>
<td>98</td>
<td>1</td>
<td>0 --- ---</td>
</tr>
<tr>
<td><strong>CMP</strong> Compare memory and accumulator</td>
<td>A -&gt; M</td>
<td>Immediate</td>
<td>CMP #Op</td>
<td>C9</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>CMP Op</td>
<td>C9</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>CMP Op,X</td>
<td>D5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>CMP Op</td>
<td>CD</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute, Y</td>
<td>CMP Op,Y</td>
<td>D9</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect,X)</td>
<td>CMP Op</td>
<td>C1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect,Y)</td>
<td>CMP Op</td>
<td>D1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>CPX</strong> Compare memory and index X</td>
<td>X -&gt; M</td>
<td>Immediate</td>
<td>CPX #Op</td>
<td>E0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>CPX Op</td>
<td>E4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>CPX Op</td>
<td>E2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>CPY</strong> Compare memory and index Y</td>
<td>Y -&gt; M</td>
<td>Immediate</td>
<td>CPY #Op</td>
<td>C0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>CPY Op</td>
<td>D4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>CPY Op</td>
<td>CC</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td><strong>DEC</strong> Decrement memory by one</td>
<td>M - 1 -&gt; M</td>
<td>Zero Page</td>
<td>DEC Op</td>
<td>C6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>DEC Op,X</td>
<td>D6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>DEC Op</td>
<td>CE</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute, Y</td>
<td>DEC Op,Y</td>
<td>DE</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td><strong>DEX</strong> Decrement index X by one</td>
<td>X - 1 -&gt; X</td>
<td>Implied</td>
<td>DEX</td>
<td>CA</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>DEY</strong> Decrement index Y by one</td>
<td>Y - 1 -&gt; Y</td>
<td>Implied</td>
<td>DEY</td>
<td>88</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
## INSTRUCTION CODES

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX OP Code</th>
<th>No. Bytes</th>
<th>&quot;P&quot; Status Reg</th>
<th>&quot;N&quot; Status Reg</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EOR</strong></td>
<td>A V M → A</td>
<td>Immediate</td>
<td>EOR k/op</td>
<td>46</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>EOR Oper</td>
<td>45</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>EOR Oper X</td>
<td>55</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>EOR Oper X</td>
<td>56</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute,Y</td>
<td>EOR Oper Y</td>
<td>57</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect X)</td>
<td>EOR (Oper X)</td>
<td>41</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect Y)</td>
<td>EOR (Oper Y)</td>
<td>51</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>INC</strong></td>
<td>M + 1 → M</td>
<td>Zero Page</td>
<td>INC Oper</td>
<td>66</td>
<td>2</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>INC Oper X</td>
<td>65</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>INC Oper X</td>
<td>64</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute,Y</td>
<td>INC Oper Y</td>
<td>63</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>INX</strong></td>
<td>X + 1 → X</td>
<td>Implied</td>
<td>INX</td>
<td>66</td>
<td>1</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td><strong>INY</strong></td>
<td>Y + 1 → Y</td>
<td>Implied</td>
<td>INY</td>
<td>68</td>
<td>1</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td><strong>JMP</strong></td>
<td>PC-1 → PCH</td>
<td>Absolute</td>
<td>JMP Oper</td>
<td>6C</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(PC-2) → PCH</td>
<td>Indirect</td>
<td>JMP (Oper)</td>
<td>6C</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>JSR</strong></td>
<td>PC-1 → PCH</td>
<td>Absolute</td>
<td>JSR Oper</td>
<td>20</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(PC-2) → PCH</td>
<td>Indirect</td>
<td>JSR (Oper)</td>
<td>20</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LDA</strong></td>
<td>M → A</td>
<td>Immediate</td>
<td>LDA k/op</td>
<td>46</td>
<td>2</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LDA Oper</td>
<td>45</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>LDA Oper X</td>
<td>55</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>LDA Oper X</td>
<td>56</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute,Y</td>
<td>LDA Oper Y</td>
<td>57</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect X)</td>
<td>LDA (Oper X)</td>
<td>41</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect Y)</td>
<td>LDA (Oper Y)</td>
<td>51</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LDX</strong></td>
<td>M + X</td>
<td>Immediate</td>
<td>LDX k/op</td>
<td>46</td>
<td>2</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LDX Oper</td>
<td>45</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>LDX Oper X</td>
<td>55</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>LDX Oper X</td>
<td>56</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute,Y</td>
<td>LDX Oper Y</td>
<td>57</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect X)</td>
<td>LDX (Oper X)</td>
<td>41</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect Y)</td>
<td>LDX (Oper Y)</td>
<td>51</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LDY</strong></td>
<td>M + Y</td>
<td>Immediate</td>
<td>LDY k/op</td>
<td>46</td>
<td>2</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LDY Oper</td>
<td>45</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>LDY Oper X</td>
<td>55</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>LDY Oper X</td>
<td>56</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute,Y</td>
<td>LDY Oper Y</td>
<td>57</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect X)</td>
<td>LDY (Oper X)</td>
<td>41</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect Y)</td>
<td>LDY (Oper Y)</td>
<td>51</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LSR</strong></td>
<td>A V M → A</td>
<td>Immediate</td>
<td>LSR k/op</td>
<td>46</td>
<td>2</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LSR Oper</td>
<td>45</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>LSR Oper X</td>
<td>55</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>LSR Oper X</td>
<td>56</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NOP</strong></td>
<td>No operation</td>
<td>No Operation</td>
<td>NOP</td>
<td>EA</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ORA</strong></td>
<td>A V M → A</td>
<td>Immediate</td>
<td>ORA k/op</td>
<td>46</td>
<td>2</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>ORA Oper</td>
<td>45</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page,X</td>
<td>ORA Oper X</td>
<td>55</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>ORA Oper X</td>
<td>56</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute,Y</td>
<td>ORA Oper Y</td>
<td>57</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect X)</td>
<td>ORA (Oper X)</td>
<td>41</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect Y)</td>
<td>ORA (Oper Y)</td>
<td>51</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PHA</strong></td>
<td>Push accumulator on stack</td>
<td></td>
<td>PHA</td>
<td>45</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PHP</strong></td>
<td>Push processor status on stack</td>
<td></td>
<td>PHP</td>
<td>68</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PLA</strong></td>
<td>Pull accumulator from stack</td>
<td></td>
<td>PLA</td>
<td>68</td>
<td>1</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td><strong>PLP</strong></td>
<td>Pull processor status from stack</td>
<td></td>
<td>PLP</td>
<td>28</td>
<td>1</td>
<td></td>
<td>From Stack</td>
</tr>
<tr>
<td><strong>ROL</strong></td>
<td>Rotate one bit left (memory or accumulator)</td>
<td></td>
<td>ROL</td>
<td>29</td>
<td>1</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(See Figure 3)</td>
<td></td>
<td>ROL (Oper X)</td>
<td>29</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ROR</strong></td>
<td>Rotate one bit right (memory or accumulator)</td>
<td></td>
<td>ROR</td>
<td>6A</td>
<td>1</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(See Figure 3)</td>
<td></td>
<td>ROR (Oper X)</td>
<td>67</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### INSTRUCTION CODES

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX OP Code</th>
<th>No. Bytes</th>
<th>P' Status Reg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTI</td>
<td>P I PC I</td>
<td>Implied</td>
<td>RTI</td>
<td>40</td>
<td>1</td>
<td>From Stack</td>
</tr>
<tr>
<td>RTS</td>
<td>PC I PC-1→PC</td>
<td>Implied</td>
<td>RTS</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBC</td>
<td>A M C A</td>
<td>Immediate</td>
<td>SBC #Oper</td>
<td>E0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A Z PC X</td>
<td>Zero Page</td>
<td>SBC Oper</td>
<td>E5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A Absolute</td>
<td>Absolute</td>
<td>SBC Oper</td>
<td>E0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A Absolute</td>
<td>Absolute</td>
<td>SBC Oper</td>
<td>F0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A (Ind X)</td>
<td>Absolute Y</td>
<td>SBC (Oper Y)</td>
<td>E1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A (Ind X)</td>
<td>Absolute Y</td>
<td>SBC (Oper X)</td>
<td>F1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>SEC</td>
<td>1 → C</td>
<td>Implied</td>
<td>SEC</td>
<td>30</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SED</td>
<td>1 → D</td>
<td>Implied</td>
<td>SED</td>
<td>30</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SEI</td>
<td>1 → I</td>
<td>Implied</td>
<td>SEI</td>
<td>78</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>STA</td>
<td>A M</td>
<td>Zero Page</td>
<td>STA Oper</td>
<td>85</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A Z PC X</td>
<td>Zero Page</td>
<td>STA Oper,X</td>
<td>95</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A Absolute</td>
<td>Absolute</td>
<td>STA Oper</td>
<td>80</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A Absolute</td>
<td>Absolute</td>
<td>STA Oper,Y</td>
<td>90</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A (Ind X)</td>
<td>Absolute Y</td>
<td>STA (Oper,X)</td>
<td>81</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A (Ind X)</td>
<td>Absolute Y</td>
<td>STA (Oper,Y)</td>
<td>91</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>STX</td>
<td>X M</td>
<td>Zero Page</td>
<td>STX Oper</td>
<td>86</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X Z PC Y</td>
<td>Zero Page</td>
<td>STX Oper,Y</td>
<td>96</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X Absolute</td>
<td>Absolute</td>
<td>STX Oper</td>
<td>86</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>STY</td>
<td>Y M</td>
<td>Zero Page</td>
<td>STY Oper</td>
<td>84</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Y Z PC X</td>
<td>Zero Page</td>
<td>STY Oper,Y</td>
<td>94</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Y Absolute</td>
<td>Absolute</td>
<td>STY Oper</td>
<td>84</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>TAX</td>
<td>A X</td>
<td>Implied</td>
<td>TAX</td>
<td>AA</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>TAY</td>
<td>A Y</td>
<td>Implied</td>
<td>TAY</td>
<td>AB</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>TSX</td>
<td>S X</td>
<td>Implied</td>
<td>TSX</td>
<td>8A</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

### Additional 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>
</tr>
</thead>
<tbody>
<tr>
<td>TXA</td>
<td>X A</td>
<td>Implied</td>
<td>TXA</td>
<td>8A</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>TXS</td>
<td>X S</td>
<td>Implied</td>
<td>TXS</td>
<td>8A</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>TYA</td>
<td>Y A</td>
<td>Implied</td>
<td>TYA</td>
<td>9A</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- **P' Status Reg.:** Indicates the status register affected by the instruction.
Example of a document page with hexadecimal operation codes.
APPLE II HARDWARE

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

INTRODUCTION

ITEMS YOU WILL NEED:

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

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

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

c. 1 ea. 2" speaker with cable.

d. 1 ea. Preliminary Manual

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

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

In addition you will need:

g. A color TV set (or B & W) equipped with a direct video input connector for best performance or a commercially available RF modulator such as a “Pixi-verter”™. Higher channel (7-13) modulators generally provide better system performance than lower channel modulators (2-6).

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

1. +12 Volts with the following current capacity!
   a. For 4K or 16K systems - 350mA.
   b. For 8K, 20K or 32K - 550mA.
   c. For 12K, 24K, 36K or 48K - 850mA.

2. +5 Volts at 1.6 amps

3. -5 Volts at 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 s queels 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 500μV to obtain a clean picture. Therefore we recommend that a good, co-ax cable be used to carry the signal from any modulator to the TV set, such as RG-59u (with copper shield), Belden #8241 or an equivalent miniature type such as Belden #8218. We also recommend that the RF modulator be closed in a tight metal box (an unpainted die cast aluminum box such as Pomona #2428). Even with these precautions, some trouble may be encountered with worms, and can be greatly helped by threading the coax cable connecting the modulator to the TV set repeatedly through a Ferrite toroid core. Apple Computer supplies these cores in a kit:along with a 4 circuit connector/cable assembly to match the auxiliary video connector found on the Apple II board. This kit has order number A2M010X. The M&R "Sup-r-Mod is supplied with a coax cable and toroids.

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

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

The ASR - 33 Teletype

The ASR - 33 Teletype of recent vintage has a transistor circuit to drive its solenoids. This circuit is quite easy to interface to, since it is provided with its own power supply. (Figure la) 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. PR#Ø will inactivate the printer transferring control back to the Video monitor as the primary output device.

3. In Monitor mode $37ØØ activates the printer and hitting the "RESET" key exits the program.

Saving the Machine Language Program

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

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

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

3. Type in "37Ø.Ø3E9W 37Ø.Ø3E9W".

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

5. When the program has been written to tape, the asterisk 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...XYZØ123456789"
   20 PR#Ø
   25 END

4. Type in RUN and hit the "RETURN" key. The text in line 15 should be printed on the teletype and control is returned to the keyboard and Video monitor
Line 10 activates the teletype machine routine and all "PRINT" statements following it will be printed to the teletype until a PR#0 statement is encountered. Then the text in line 15 will appear on the teletype's output. Line 20 deactivates the printer and the program ends on line 25.

Conclusion

With the circuits and machine language program described in this paper the user may develop a relatively simple serial output interface to an ASR-3 or RS-232 compatible printers. This circuit can be activated through BASIC or monitor modes. And is a valuable addition to any users program library.
FIGURE 2  RS-232

3.3K  

2N3906 (OR EQUIV.)

+12 (JUMPERED TO +12 SUPPLY)

PIN 15  J-14

3.3K  

RESISTORS ARE 1/4 WATT CARBON

118
TELETYPETE DRIVER ROUTINES

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

**WARNING: OPERAND OVERFLOW IN LINE 27**

**TTINIT:**

LDA #$72 ;POINT TO TTY ROUTINES
STA CSWL
LDA #$7F8 ;COLUMN COUNT LOC.
LDA CH
STA COLCNT ;WHERE WE ARE NOW.
RTS
PHA ;SAVE TWICE
PHA ;ON STACK.
LDA COLCNT ;CHECK FOR A TAB.
CMP CH
PLA ;RESTORE OUTPUT CHAR.
BCS TESTCTRL ;IF C SET, NO TAB
PHA
LDA #$80 ;PRINT A SPACE.
TESTCTRL:BIT RTS1 ;TRICK TO DETERMINE
BEQ PRNTIT ;IF CONTROL CHAR.
INC COLCNT ;IF NOT, ADD ONE TO CM
JSR DOCHAR ;PRINT THE CHAR ON TTY
PLA ;RESTORE CHAR
PHA TTOUT2 ;AND PUT BACK ON STAC
BCC #$0D ;DO MORE SPACES FOR TA
FOR A ;CHECK FOR CAR RET.
ASL FINISH ;ELIM PARITY
BNE ;IF NOT CR, DONE.
**TELETYPe DRIVER ROUTINES**

---

**FINISH:**

```
; CLEAR COLUMN COUNT
STA COLCNT

; NOW DO LINE FEED
LDA #38A
JSR DOCHAR

; CHECK IF IN MARGIN
JSR 7AIT

; FOR CR, RESET CH
S3C 7V7DTH

; IF SO, CARRY SET.
BCC RETURN

; ADJUST CH
ADC #11F

; SET CH
STA CH
```

---

**SETCHAN**

```
; SAVE STATUS.
PHP YSAVE

; 11 BITS (1 START, 1 2
BEGIN 7TH SPACE (ST2

; SAVE A REG AND SET FOI
PHA #SOS

; SEND A SPACE
TTOUT3: LDA SPACE

; SEND A MARK
TTOUT4: LDA MARK

; DELAY 9.091 MSEC FOR
DLY1: LSR #$20

; NEXT BIT (STOP BITS ?
DLY2: BCC A

; LOOP 11 3ITS.
PLA DLY1

; RESTORE Y-REG.
ROR

; RESTORE STATUS
DEY A

; RETURN
```

---

**MARKOUT**

```
; SAVE A REG AND SET FOI
PHA #D7

; DELAY 9.091 MSEC FOR
TTOUT4: LDA MARKOUT

; SEND A SPACE
33C SPACE

; SEND A MARK
LDA TTOUT4

; 110 BAUD
BNE LOOP 11 3ITS.

; NEXT BIT (STOP BITS ?
3NE #101

; RESTORE STATUS
SBC

; RESTORE Y-REG.
PLA YSAVE

; RETURN
```

---

**SUCCESSFUL ASSEMBLY: NO ERRORS**

---

**Figure 3b**
CROSS-REFERENCE: TELETYPES DRIVER ROUTINES

CH 0024 0033 0039 0065
CLOCNT 0718 0034 0038 0046 0054 0059
0SYL 0036 0028 0030
DLYI 0305 0085
DLY2 0308 0082
DOCHAR 0301 0047 0056
FINISH 0330 0053
MARK CO58 0077
MARKOUT 0300 0074
PRNTIT 0397 0045
RETURN 038F 0063
RTS1 0300 0044
SETCH 0330 0060
SPACE CO59 0075
TESTCTRL 033F 0041
TTINIT 0370
TTOUT 0332 0027 0029
TTOUT2 0384 0050
TTOUT3 03C8 0089
TTOUT4 0303 0076
WAIT FCAB 0058
WNDWDTH 0021 0032 0061
YSAVE 0778 0069 0090
ILE:

FIGURE 3c
INTERFACING THE APPLE

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

TABLE OF CONTENTS

1. Connector Location Diagram
2. Cassette Data Jacks (2 Each)
3. Game I/O Connector
4. Keyboard Connector
5. Peripheral Connectors (8 Each)
6. Power Connector
7. Speaker Connector
8. Video Output Jack
9. Auxiliary Video Output Connector
Figure 1A  APPLE II Board-Complete View
Figure 1B Connector Location Detail

APPLE II PC BOARD

Front of PC Board

Right Side of PC Board

BACK EDGE OF PC BOARD

KEYBOARD CONNECTOR

POWER CONNECTOR

TOP VIEW

PERIPHERALS

VIDEO OUTPUT

GAME I/O

SPEAKER CONNECTOR

CONNECTION LOCATIONS

CASSETTE DATA IN

CASSETTE DATA OUT

K1 K13 K14

K12

J14 J14B

0 1 2 3 4 5 6 7

J2 J4 J5 J6 J8 J9 J11 J12
CASSETTE JACKS

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

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

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

GAME I/O CONNECTOR

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

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 Ø2 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 Ø-150K ohm variable resistance and +5V for each paddle. Internal 100 ohm resistors are provided in series with external pot to prevent excess current if pot goes completely to zero ohms.

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

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

KEYBOARD CONNECTOR

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

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

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

NC: No connection.

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

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

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

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

**PERIPHERAL CONNECTORS**

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

**SIGNAL DESCRIPTION FOR PERIPHERAL I/O**

AO-A15: 16 bit system address bus. Addresses are set up by the 6502 within 300nS after the beginning of 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 total low power schottky TTL loads.
**DMA:** Direct Memory Access control output. This line has a 3K Ohm pullup to +5V and should be driven with an open collector output.

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

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

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

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

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

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

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

**I/O STROBE:** Pin 20 on all peripheral connectors will go low during Ø, of a read or write to any address C800-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 0:** Microprocessor phase V clock. Will drive up to a total of 16 standard TTL loads.

**Ø 1:** Phase 1 clock, complement of Ø 0. Will drive up to a total of 16 standard TTL loads.

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

**+12V:** Positive 12-Volt supply.

**+5V:** Positive 5-Volt supply

**-5V:** Negative 5-Volt supply.

**-12V:** Negative 12-Volt supply.

---

**POWER CONNECTOR**

The four voltages required by the APPLE II are supplied via this AMP #9-35028-1,6 pin connector. See location and pin out in Figures 1 and 5.

**PIN DESCRIPTION**

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

- GND
- DMA IN
- INT IN
- NMI
- IRQ
- RES
- INH
- -12V
- -5V
- N.C.
- 7M
- Q3
- USER 1
- DEVICE SELECT
- D7
- D6
- D5
- D4
- D3
- D2
- D1
- D0
- +12V

(LOW EDGE OF PC BOARD)
LOCATIONS J2 TO J12

Figure 5

POWER CONNECTOR

PINOUT
(TOP VIEW)

(TOWARD RIGHT SIDE OF PC BOARD)

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

LOCATION K1
SPEAKER CONNECTOR

This is a MOLEX KK 100 series connector with two .25" square pins on .1" centers. See location and pin out in Figures 1 and 6.

SIGNAL DESCRIPTION FOR SPEAKER

+5V: System +5 Volts

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

VIDEO OUTPUT JACK

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

A video level control near the connector allows the output level to be adjusted from 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

+12V

-5V

VIDEO

GND

Right Edge of PC Board

Back Edge of PC Board

LOCATION J14B
INSTALLING YOUR OWN RAM

THE POSSIBILITIES

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

RESTRICTIONS

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

1. All sockets in a row must have the same type (4K or 16K) RAMs.

2. There MUST be RAM assigned to the zero block of addresses.

ASSIGNING RAM

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

Figure 8

MEMORY SELECT SOCKETS

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

Let us learn by example:

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>DECIMAL</th>
<th>Hexadecimal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
<td>text and color graphics display pages, 8502 stack, pointers, etc.</td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
<td>high res. graphics display primary page</td>
</tr>
<tr>
<td>3000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td></td>
<td>high res. graphics display secondary page</td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8000</td>
<td></td>
<td>addresses dedicated to hardware functions</td>
</tr>
<tr>
<td>9000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Memory Map Pages 0 to BFF

<table>
<thead>
<tr>
<th>PAGE ZERO</th>
<th>ADDRESS(ES)</th>
<th>USED BY</th>
<th>USED FOR</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000-001F</td>
<td>0000-024F</td>
<td>UTILITY</td>
<td>register area for &quot;sweet 16&quot; 16 bit firmware processor.</td>
<td></td>
</tr>
<tr>
<td>0020-004D</td>
<td>004E-006F</td>
<td>MONITOR</td>
<td>holds a 16 bit number that is randomized with each key entry.</td>
<td></td>
</tr>
<tr>
<td>0050-0055</td>
<td>0055-00FF</td>
<td>UTILITY</td>
<td>integer multiply and divide work space.</td>
<td></td>
</tr>
<tr>
<td>0055-00FF</td>
<td>0060-00FF</td>
<td>BASIC</td>
<td>floatpoint work space.</td>
<td></td>
</tr>
<tr>
<td>0070-00FF</td>
<td>0100-010F</td>
<td>UTILITY</td>
<td>subroutine return stack.</td>
<td></td>
</tr>
<tr>
<td>0100-01FF</td>
<td>0150-025F</td>
<td></td>
<td>character input buffer.</td>
<td></td>
</tr>
<tr>
<td>PAGE ONE</td>
<td>0200-02FF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAGE TWO</td>
<td>0300-03FF</td>
<td>MONITOR</td>
<td>Y (control Y) will cause a JSR to this location.</td>
<td></td>
</tr>
<tr>
<td>PAGE THREE</td>
<td>03F8</td>
<td></td>
<td>WMI's are vectored to this location.</td>
<td></td>
</tr>
<tr>
<td>03F8</td>
<td></td>
<td>DISPLAY</td>
<td>IRQ's are vectored to the address pointed to by these locations.</td>
<td></td>
</tr>
<tr>
<td>03FE-03FF</td>
<td></td>
<td>DISPLAY</td>
<td>text or color graphics primary page.</td>
<td>BASIC initializes LQEMM to location 0800.</td>
</tr>
<tr>
<td>0400-07FF</td>
<td></td>
<td>DISPLAY</td>
<td>text or color graphics secondary page.</td>
<td></td>
</tr>
<tr>
<td>0800-0BFF</td>
<td></td>
<td>DISPLAY</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

136
<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></td>
<td></td>
<td>State of Switch 1 ( \land ) Game I/O connector appears in bit 7.</td>
</tr>
<tr>
<td>C061/9</td>
<td>&quot;SW1&quot;</td>
<td>State of Switch 2 input on Game I/O connector appears in bit 7.</td>
</tr>
<tr>
<td>C062/A</td>
<td>&quot;SW2&quot;</td>
<td>State of Switch 3 input on Game I/O connector appears in bit 7.</td>
</tr>
<tr>
<td>C063/B</td>
<td>&quot;SW3&quot;</td>
<td>State of timer output for Paddle 0 appears in bit 7.</td>
</tr>
<tr>
<td>C064/C</td>
<td>Paddle 0 timer output</td>
<td>State of timer output for Paddle 1 appears in bit 7.</td>
</tr>
<tr>
<td>C065/D</td>
<td>&quot;1&quot;</td>
<td>State of timer output for Paddle 2 appears in bit 7.</td>
</tr>
<tr>
<td>C066/E</td>
<td>&quot;2&quot;</td>
<td>State of timer output for Paddle 3 appears in bit 7.</td>
</tr>
<tr>
<td>C067/F</td>
<td>&quot;3&quot;</td>
<td>Triggers paddle timers during ( \varphi_2 ).</td>
</tr>
<tr>
<td>C07X</td>
<td>&quot;PDL STB&quot;</td>
<td>Pin 41 on the selected Peripheral Connector goes low during ( \varphi_2 ).</td>
</tr>
<tr>
<td>C08X</td>
<td>DEVICE SELECT 0</td>
<td></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>Expansion connectors.</td>
</tr>
<tr>
<td>C11X</td>
<td>&quot;9&quot;</td>
<td></td>
</tr>
<tr>
<td>C12X</td>
<td>&quot;A&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>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>Expansion connectors.</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-FFFF</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, 3.530 MHz.

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

01: Microprocessor phase 1 clock, complement of 00, 1.023 Mhz nominal.

02: Same as 00. Included here because the 6502 hardware and programming manuals use the designation 02 instead of 00.

03: 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 01, and is stable about 300nS after the start of 01.

DATA WRITE: During a write cycle, data from the microprocessor appears on the data bus during 02, and is stable about 300nS after the start of 02.

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 02.

SYSTEM TIMING DIAGRAM
FIGURE S-1  APPLE II SYSTEM DIAGRAM
FIGURE S-2  MPU AND SYSTEM BUS
FIGURE S-3  REFERENCE OSCILLATOR AND SYSTEM TIMING
FIGURE S-4  SYNC COUNTER
FIGURE S-5 ROM MEMORY
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-10 ON-BOARD I/O
FIGURE S-11 VIDEO GENERATOR