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

Any donations for better software and hardware, what I have is old and slow, can be made to marketplace@seaside.ns.ca, Thanks!

iDonate

PayPal

Current Donators to this project are:
Dan Chisarick
APPLE II
Reference Manual
January 1978
APPLE II Reference Manual

TABLE OF CONTENTS

A. GETTING STARTED WITH YOUR APPLE II
   1. Unpacking 1
   2. Warranty Registration Card 1
   3. Check for Shipping Damage 2
   4. Power Up 2
   5. APPLE II Speaks Several Languages 2
   6. APPLE Integer BASIC 3
   7. Running Your First and Second Programs 3
   8. Running 16K Starrek 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 Starrek 14
  13. Loading HIRES Demo Tape 15

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

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

D. APPLE II HARDWARE
   1. Getting Started with Your APPLE II Board 107
   2. APPLE II Switching Power Supply 112
   3. Interfacing with the Home TV 114
   4. Simple Serial Output 115
   5. Interfacing the APPLE - Signals, Loading, Pin Connections 122
   6. Memory Options, Expansion, Map, Address 133
   7. System Timing 140
   8. Schematics 141

13. Additional BASIC Program Examples 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
GETTING STARTED WITH YOUR APPLE II

Unpacking

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

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

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

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

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

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

Warranty Registration Card

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

A2S00MMX

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

A2S0008X

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

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

Power Up

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

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

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

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

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

Apple Integer BASIC

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

Running Your First and Second Program

Read through the next three sections that include:

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

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

Running 16K Startrek

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

INTRODUCTION

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

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

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

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

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

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

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

SYSTEM CHECKOUT

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

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

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

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

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

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

1. **Hit the RESET key.**  
   An asterisk, "+", 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 asterisk.

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 preceeding is followed a good tape load will result.

UNSOLVED PROBLEMS

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

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

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

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

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

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

COLOR DEMO TAPE

PROGRAM DESCRIPTION

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

REQUIREMENTS

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

5 GOTO 15
10 0< (PDL (0)-20)/6: IF 0<0 THEN
0: IF 0<34 THEN 0=34: COLOR=D:
VLIN 0,0+5 AT 0: COLOR=A:
IF P>0 THEN 175: IF 0 THEN
VLIN 0,0-1 AT 0: P=0: 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):
VLIN 0,39 AT I
30 NEXT I: POKE 34,20: PRINT :
PRINT: FOR I=0 TO 15: VTAB 21+I MOD 2:
TAB I+I+1: PRINT I;: NEXT I:
POKE 34,22: YTAB 24: PRINT
'BACKGROUND';
35 GOSUB 95:AHE: PRINT 'EVEN BRICK'
: GOSUB 95:EHE: PRINT 'ODD BRICK'
K$: GOSUB 95:EHE: PRINT 'PADDLE'
: GOSUB 95:DHE: PRINT 'BALL'
: GOSUB 95
40 POKE 34,29: COLOR=A: FOR I=
0 TO 39: VLIN 0,39 AT I: NEXT
I: FOR I=20 TO 34 STEP 2: TAB
I=1: PRINT I/2-9: COLOR=0:
VLIN 0,39 AT I: COLOR=C: FOR
J=I MOD 4 TO 39 STEP 4
45 VLIN J,J+1 AT I: NEXT J, I: TAB
S: PRINT 'SCORE =': PRINT
 : FOR I=1 TO 30: S=S+10: P=
S:=S+10: S:=S+10:
50 COLOR=A: PLOT K,Y/S;X=I;Y=I:
RND (128): X=I=1: RND (5):
2: L=L-1: IF ('L THEN 120: TAB
6: IF I=1 THEN PRINT L; 'BALLS
LEFT':
55 IF L=1 THEN PRINT 'LAST BALL, '
: A$: PRINT: FOR I=1 TO 30:
: GOSUB 16: NEXT I: H=I
60 J=Y/I: IF I>0 AND I<120 THEN
65 I=I+4; J=J: FOR I=1 TO 16:
: K: PEEK (-16336)-PEEK (-16336)
70 Z=PEEK (-16336)-PEEK (-16336)
: GOTO 85
75 FOR I=1 TO 6: M=PEEK (-16336)
: GOTO 60
80 X=I: M=0
85 PLOT X,Y/3: COLOR=E: PLOT I,
K=I+1=1: GOTO 60
90 PRINT 'INVALID, REENTER';
95 INPUT 'COLOR (0 TO 15)', E:
: IF E<0 OR E>15 THEN 90: RETURN
100 IF M THEN V=ABS (V): VLIN
K/2+X,K/2+Y AT I=1: S=S+1: I=2:
9: VTAB 21: TAB 13: PRINT S
105 ON PEEK (-16336)-PEEK (-16336)
: GOTO 160
110 IF S<720 THEN 80
115 PRINT 'CONGRATULATIONS, ': A$
: Y 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
150 PRINT 'EXCELLENT. ': GOTO 165
155 PRINT 'NEARLY PERFECT.'
160 PRINT 'HEARTLY PERFECT. '
165 PRINT 'ANOTHER GAME '? A$: '
: INPUT A$: IF A$(1,1)="Y" THEN
25 23: TEXT: CALL -936:
VTAB 10: TAB 10: PRINT 'GAME OV
ER': END
170 ON PDL (0)-20)/6: IF 0<0 THEN
0: IF 0<34 THEN 0=34: COLOR=D:
VLIN 0,0+5 AT 0: COLOR=A:
IF P>0 THEN 175: IF 0 THEN
VLIN 0,0-1 AT 0: P=0: RETURN
175 IF P=0 THEN RETURN: IF 0<34
THEN VLIN 0+6,39 AT 0: P=0: RETURN
180 FOR I=1 TO 80: IF PEEK (-16336)
: NEXT I: GOTO 50
COLOR DEMO PROGRAM LISTING

10 DIM C(43): POKE 2,173: POKE 3,43: POKE 4,192: POKE 5,155:
6,0: POKE 7,32: POKE 8,168: POKE 9,252: POKE 10,165: POKE 11,0: POKE 12,200:
POKE 20,76: POKE 21,2: POKE 22,0: POKE 23,96
30 TEXT TO CALL -936: VTab 4: TAB 8: PRINT "4K COLOR DEMOS": PRINT
: PRINT "1 LINES": PRINT "2 CROSS": PRINT "3 WEAVING": PRINT "4 TUNNEL": PRINT "5 CIRCLE": PRINT "6 SPIRAL": PRINT
"7 TONES": PRINT "8 SPRING": PRINT "9 HYPERBOLA": PRINT
"10 COLOR BARS": PRINT "PRINT "** NEEDS POL(0) CONNECTED"": PRINT
60 PRINT "HIT ANY KEY FOR NEW DEMO": PRINT 2:0: PRINT: INPUT "WHICH DEMO?"
#": I: GR: IF I<0 AND I<11 THEN GOTO 100:1: GOTO 36
70 INPUT "WHICH DEMO WOULD YOU LIKE"
", I: IF I>1 AND I<20 THEN GOTO 100:1: GOTO 36
100 I=I+1 MOD 79: J=I*(34996)*79
-I-J: GOSUB 2000: GOSUB 10000:
: GOTO 100
200 I=I+1 MOD 39: J=I: GOSUB 2000:
: J=J+1: GOSUB 2000: GOSUB 10000: GOTO 200
300 J=J+1: I MOD 22+1: FOR I=1 TO 1295:
COLOR=I MOD 7: PLOT (2*1) MOD 37,(3+1) MOD 35:
NEXT I: GOSUB 10000: GOTO 300
400 FOR I=1 TO 4: C(I)= RND (16):
: NEXT I
410 FOR I=3 TO 1 STEP -1 (1+1):
C(I): NEXT I C(I)= RND (16):
: FOR I=1 TO 5: FOR I=1 TO 4
420 COLOR=C(I): L=J*5+14-I:X=3:-
L: HLIN K,L AT K: VLIN K,L AT L:
L: HLIN K,L AT L: VLIN K,L AT K:
: NEXT J,1: GOSUB 10000: GOTO 410
500 2=2+B: GOTO 900
600 COLOR= RND (16): FOR 1=0 TO 18 STEP 2: I=39-1: HLIN I,J AT I:
GOSUB 648: VLIN I,J AT J:
GOSUB 648
610 HLIN I=2,J AT J: GOSUB 648:
VLIN I=2,J AT J=2: GOSUB 648:
: NEXT I
620 COLOR= RND (16): FOR 1=18 TO 0 STEP -2: I=39-1: HLIN I,J:
J AT I=2: GOSUB 648: HLIN I+2,J AT J=2: GOSUB 648
630 VLIN I,J AT J: GOSUB 648: HLIN I,J AT I=2: GOSUB 648:
GOSUB 10000: GOTO 600
700 I=RND (30)+3: J=I+4+1+4:26+
70: X=3: RND (7): <PDL 0,10>:
POKE 6,1: POKE 1,4: MOD 256:
POKE 24, (K*255)+1: CALL 2
GOSUB 10000: GOTO 780
680 X=3: R=10000: P=3+10: X=4:Y=0
J=1: COLOR=6: HLIN 0,39 AT 4:
COLOR=9: GOSUB 880: COLOR=12:
VLI N 5, M AT X
800 N=2:A.P. -Y: W: COLOR=0: GOSUB
880: VLIN 5,39 AT X: X=N+1: IF
X=39 THEN 680: X=5: VLIN 5,39
AT I=1: VLIN 5,39 AT 2
820 P=R+A: N=1: Y=100: COLOR=12: GOSUB
880: COLOR=9: VLIN 5, M AT X;
COLOR=15: PLOT X, -X: FOR
I=0 TO J: NEXT I: GOSUB 10000:
GOTO 810
890 M=L, X=1=L, X=1=L, X=1=L
VLIN 1, L2 AT X=1, VLIN L1, L2 AT X:
VLI N L1, L2 AT X=1: RETURN
900 I=1:1 MOD 15: FOR Y=0 TO 39:
FOR X=0 TO 39: COLOR=I+X ABS
(28-X)*X:ABS (28-Y)*1: Z:
PLOT X,Y: NEXT X,Y: GOSUB
10000: GOTO 900
1000 CALL -936
1010 J=I+I MOD 32: COLOR=J=2: YLIN
8,39 AT 3+3: VTab 24+(J/2) MOD
2: TAB 3+J: IF J MOD 2 THEN
PRINT J, 1: GOSUB 10000: GOTO
1010
2000 COLOR=RND (16): HLIN 0,39 AT
J: COLOR=RND (16): VLIN 0, 39 AT J: RETURN
10000 IF PEER (-16334)*128 THEN RETURN
: POKE -16334,0: POP 0: 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 DESTINATION.

B. A 2 REGENERATES THE ENERGY AT THE EXPENSE OF TIME.

C. A 3 GIVES THE CONTENTS OF THE IMMEDIATE ADJACENT QUADRANTS. THE GALAXY IS WRAP-AROUND IN ALL DIRECTIONS.

D. A 4 FIRES PHASERS AT THE EXPENSE OF AVAILABLE ENERGY.

E. A 5 INITIATES A SET OF QUESTIONS FOR TORPEDO FIRING. THEY CAN BE FIRED AUTOMATICALLY IF THEY HAVE BEEN LOCKED ON TARGET WHILE IN THE COMPUTER MODE, OR MAY BE FIRED MANUALLY IF THE TRAJECTORY ANGLE IS KNOWN.

F. A 6, 8 AND 10 ALL GIVE INFORMATION ABOUT THE STATUS OF THE SHIP AND ITS ENVIRONMENT.

G. A 9 SETS THE SHIELD ENERGY/AVAILABLE ENERGY RATIO.


I. A 7 ENTERS A COMPUTER WHICH WILL RESPOND TO THE FOLLOWING INSTRUCTIONS:

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

IN THE FIRST FIVE ONE WILL HAVE TO GIVE COORDINATES. COORDINATES ARE GIVEN IN MATHEMATICAL NOTATION WITH THE EXCEPTION THAT THE 'Y' VALUE IS GIVEN FIRST. AN EXAMPLE WOULD BE 'Y,X' COURSE OR TRAJECTORY:

0
|
270-----------------------90
|
180

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

PROCEDURE

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

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

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

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

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

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

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

NEW
Clears (Scratches) current BASIC program.

NO_DSP var
Clears DSP mode for variable var.

NO TRACE
Clears TRACE mode.

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Sample Statement</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prefix Operators</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( )</td>
<td>10 X = 4*(5 + X)</td>
<td>Expressions within parenthesis ( ) are always evaluated first.</td>
</tr>
<tr>
<td>+</td>
<td>20 X = 1+4*5</td>
<td>Optional; +1 times following expression.</td>
</tr>
<tr>
<td>-</td>
<td>30 ALPHA = -(BETA +2)</td>
<td>Negation of following expression.</td>
</tr>
<tr>
<td>NOT</td>
<td>40 IF A NOT B THEN 200</td>
<td>Logical Negation of following expression; 0 if expression is true (non-zero), 1 if expression is false (zero).</td>
</tr>
</tbody>
</table>

| **Arithmetic Operators** | | |
| † | 60 Y = X ^ 3 | Exponentiate as in \(X^3\). NOTE: † is shifted letter N. |
| * | 70 LET DOTS=A*B*N2 | Multiplication. NOTE: Implied multiplication such as \((2 + 3)(4)\) is not allowed thus N2 in example is a variable not \(N \times 2\). |
| / | 80 PRINT GAMMA/S | Divide |
| MOD | 90 X = 12 MOD 7 | Modulo: Remainder after division of first expression by second expression. |
| + | 100 X = X MOD(Y+2) | |
| - | 110 P = L + G | Add |
| - | 120 XY4 = H-D | Substract |
| = | 130 HEIGHT=15 | Assignment operator; assigns a value to a variable. LET is optional |
| 140 LET SIZE=7*5 | |
| 150 A(8) = 2 | |
| 155 ALPHA$ = "PLEASE" | |
Relational and Logical Operators

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Sample Statement</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>160 IF D = E THEN 500</td>
<td>Expression &quot;equals&quot; expression.</td>
</tr>
<tr>
<td>=</td>
<td>170 IF A$(1,1)=&quot;Y&quot; THEN 5VV</td>
<td>String variable &quot;equal&quot; string variable.</td>
</tr>
<tr>
<td># or &lt; &gt;</td>
<td>180 IF ALPHA #X*Y THEN 500</td>
<td>Expression &quot;does not equal&quot; expression.</td>
</tr>
<tr>
<td>#</td>
<td>190 IF A$ # &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</th>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(expr)</td>
<td>300 PRINT ABS(X)</td>
<td>Gives absolute value of the expression expr.</td>
</tr>
<tr>
<td>ASC(str$)</td>
<td>310 PRINT ASC(&quot;BACK&quot;)</td>
<td>Gives decimal ASCII value of designated string variable str. If more than one character is in designated string or sub-string, it gives decimal ASCII value of first character.</td>
</tr>
<tr>
<td>LEN(str$)</td>
<td>340 PRINT LEN(B$)</td>
<td>Gives current length of designated string variable str$; i.e., number of characters.</td>
</tr>
<tr>
<td>PDL(expr)</td>
<td>350 PRINT PDL(X)</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>PEEK(expr)</td>
<td>360 PRINT PEEK(X)</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>RND(expr)</td>
<td>370 PRINT RND(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>SCRN(expr1, expr2)</td>
<td>380 PRINT SCRN (X1,Y1)</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 standar mixed colorgraphics display mode as set by GR command or Ø to 47 if in all color mode set by POKE -163Ø4 ,Ø: POKE -163Ø2,Ø'.</td>
</tr>
<tr>
<td>SGN(expr)</td>
<td>390 PRINT SGN(X)</td>
<td>Gives 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 alpha-numeric characters up to 100. Variable names may not contain buried any of the following words: AND, AT, MOD, OR, STEP, or THEN. Variable names may not begin with the letters END, LET, or REM. String variables names must end with a $ (dollar sign). Multiple statements may appear under the same line number if separated by a : (colon) as long as the total number of characters in the line (including spaces) is less than approximately 150 characters. Most statements may also be used as commands. BASIC statements are executed by RUN or GOTO commands.

NAME

CALL expr 10 CALL-936 Causes execution of a machine level language subroutine at decimal memory location specified by expression expr. Locations above 32767 are specified using negative numbers; i.e., location in example 10 is hexadecimal number $FC53.

COLOR=expr 30 COLOR=12 In standard resolution color (GR) graphics mode, this command sets screen TV color to value in expression expr in the range 0 to 15 as described in Table A. Actually expression expr may be in the range 0 to 255 without error message since it is implemented as if it were expression expr MOD 16.

DIM vari(expr1) 50 DIM A(20),B(10) 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 redefined at any time; thus, example in line is illegal but 85 is allowed.

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

var$ (expr3) 70 DIM C (2)

Illegal:

80 DIM A(30)

Legal:

85 DIM C(1000)

DSP var

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 var each time it changes and the line number where the change occurred.
<table>
<thead>
<tr>
<th>NAME</th>
<th>EXAMPLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>END</td>
<td>110 END</td>
<td>Stops program execution. Sends carriage return and &quot;&gt; &quot; BASIC prompt) to screen.</td>
</tr>
<tr>
<td>FOR</td>
<td>110 FOR L=0 TO 39</td>
<td>Begins FOR...NEXT loop, initializes variable var to value of expression expr1 then increments it by amount in expression expr3 each time the corresponding &quot;NEXT&quot; statement is encountered, until value of expression expr2 is reached. If STEP expr3 is omitted, a STEP of +1 is assumed. Negative numbers are allowed.</td>
</tr>
<tr>
<td></td>
<td>120 FOR X=Y1 TO Y3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130 FOR 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 = Ø (Black) for top 40x40 of screen and sets scrolling window to lines 21 through 24 by 40 characters for four lines of text at bottom of screen. Example 190 sets all color mode (40x48 field) with no text at bottom of screen.</td>
</tr>
<tr>
<td></td>
<td>190 GR: POKE -163Ø2,Ø</td>
<td></td>
</tr>
<tr>
<td>HLIN</td>
<td>200 HLIN 0,39 AT 20</td>
<td>In standard resolution color graphics mode, this command draws a horizontal line of a predefined color (set by COLOR=) starting at horizontal position defined by expression expr1 and ending at position expr2 at vertical position defined by expression expr3. expr1 and expr2 must be in the range of 0 to 39 and expr1 &lt; = expr2 . expr3 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 Z,Z+6 AT 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
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 by expression *expr*. Slot Ø is not addressable from BASIC. IN#Ø (Example 33Ø) is used to return data source from peripheral I/O to keyboard connector.

**LET** Assignment operator. "LET" is optional

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

**NEXT var1, var2** Increments corresponding "FOR" variable and loops back to statement following "FOR" until variable exceeds limit.

**NO DSP var** Turns-off DSP debug mode for variable

**NO TRACE** Turns-off TRACE debug mode
In standard resolution color graphics, this command plots a small square of a predefined color (set by \texttt{COLOR=}) at horizontal location specified by expression \texttt{expr1} in range 0 to 39 and vertical location specified by expression \texttt{expr2} in range 0 to 39 (or 0 to 47 if in all graphics mode). \texttt{NOTE}: \texttt{PLOT 0, 0} is upper left and \texttt{PLOT 39, 39} (or \texttt{PLOT 39, 47}) is lower right corner.

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

"POPS" nested GOSUB return stack address by one.

Outputs data specified by variable \texttt{var} or string variable \texttt{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.

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

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

Causes branch to statement following last GOSUB; i.e., \texttt{RETURN} ends a subroutine. Do not confuse "\texttt{RETURN}" statement with Return key on keyboard.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TAB expr</strong></td>
<td>Moves cursor to absolute horizontal position specified by expression <code>expr</code> 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 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 expr1, expr2 AT expr3</strong></td>
<td>Similar to HLIN except draws vertical line starting at <code>expr1</code> and ending at <code>expr2</code> at horizontal position <code>expr3</code>.</td>
</tr>
<tr>
<td><strong>VTAB expr</strong></td>
<td>Similar to TAB. Moves cursor to absolute vertical position specified by expression <code>expr</code> in the range 1 to 24. VTAB 1 is top line on screen; VTAB 24 is bottom.</td>
</tr>
</tbody>
</table>
"Control" characters are indicated by a super-scripted "C" such as $G_C$. They are obtained by holding down the CTRL key while typing the letter. Control characters are NOT displayed on the TV screen. B and C must be followed by a carriage return. Screen editing characters are indicated by a sub-scripted "E" such as $D_E$. They are obtained by pressing and releasing the ESC key then typing specified letter. Edit characters send information only to display screen and does not send data to memory. For example, $U_C$ moves to cursor to right and copies text while $A_E$ moves cursor to right but does not copy text.

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

A_E Move cursor to right
B_E Move cursor to left
C_E Move cursor down
D_E Move cursor up
E_E Clear text from cursor to end of line
F_E Clear text from cursor to end of page
@_E Home cursor to top of page, clear text to end of page.

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.

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

<table>
<thead>
<tr>
<th>Hex</th>
<th>BASIC Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Display Mode Controls</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>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>Ø020</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>Ø021</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>Ø022</td>
<td>110 POKE 34,11</td>
<td>Set window top to line specified by T1 in range of 0 to 23</td>
</tr>
<tr>
<td>Ø023</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>Ø024</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>Ø025</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>Ø032</td>
<td>190 POKE 50,127</td>
<td>Set inverse flag if 127 (Ex. 190)</td>
</tr>
<tr>
<td></td>
<td>200 POKE 50,255</td>
<td>Set normal flag if 255(Ex. 200)</td>
</tr>
<tr>
<td>FC58</td>
<td>210 CALL -936</td>
<td>(@E) Home cursor, clear screen</td>
</tr>
<tr>
<td>FC42</td>
<td>220 CALL -958</td>
<td>(FE) Clear from cursor to end of page</td>
</tr>
<tr>
<td>Hex</td>
<td>BASIC Example</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>---------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>FC9C</td>
<td>230 CALL -868</td>
<td>(EE) Clear from cursor to end of line</td>
</tr>
<tr>
<td>FC66</td>
<td>240 CALL -922</td>
<td>(JC) Line feed</td>
</tr>
<tr>
<td>FC70</td>
<td>250 CALL -912</td>
<td>Scroll up text one line</td>
</tr>
</tbody>
</table>

**Miscellaneous**

<table>
<thead>
<tr>
<th>Hex</th>
<th>Instruction</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
C600  48K  Peripheral I/O

XX    40X   User specified RAM memory size
(HIMEM:)

XX    33X   User Workspace
(LOMEM:)

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

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

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>h1</td>
<td>h2</td>
<td>h3</td>
<td></td>
<td></td>
<td></td>
<td>h</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 1. Variable A is defined as equal to -1 (step 1). Then for convenience another variable - B - is defined as equal to Ø (step 2). Now that the variable table has been defined use of statement 2a indicates that CM is equal to 2060 (step 3). LOMEN has not been readjusted so it is equal to 2048. Therefore the variable table resides in memory from 2048 ($800$ hex) to 2060 ($88C$). Depressing the "RESET" key places the Apple II into the monitor mode (step 4).

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

READ/SAVE ROUTINE

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

The variables used in this subroutine are defined as follows:

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

SAVING A DATA TABLE

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

READING A DATA TABLE

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

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

CONCLUSION

Reading and Saving data in the format of a variable table is a relatively straightforward 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

$800.80C rewritten with labelling
FIGURE 4b

READ/SAVE PROGRAM

Ø A=Ø

10 GOTO 100

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

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


26 PRINT "DATA TABLE SAVED": RETURN

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

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


36 PRINT "DATA READ IN": RETURN

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

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

Define variable B=Ø, then hit RETURN

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

Hit the RESET key, Apple moves into Monitor mode.

Type in VARIABLE TABLE RANGE and HIT the RETURN KEY.

Example 1

Computer responds with:

Ø8ØØ- Cl Ø0 Ø8 FF FF C2 Ø0
Ø8Ø8 ØC Ø8 Ø0 Ø0 Ø0
Example 2

```plaintext
>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
  24 POKE 60,LM MOD 256: POKE 61  
    ,LM/256: POKE 62,CM MOD 256   
    : POKE 63, CM/256: CALL -307
  26 RETURN
  30 REM READ DATA SUBROUTINE
  32 POKE 60,4: POKE 61,8: POKE     
    62,5: POKE 63,8: CALL -259
  34 IF A<0 THEN 38: P=LM+A: IF P>
    HM THEN 38: CM=P: POKE 60,LM MOD
    256: POKE 61,LM/256: POKE 62  
    ,CM MOD 256: POKE 63, CM/256 
    : CALL -259
  36 RETURN
  38 PRINT **** TOO MUCH DATA BASE **
   **:END
  100 DIM A$(1), X(20)
  105 FOR I=1 TO 20: X(I)=I: NEXT  I
  110 PRINT '20 NUMBERS GENERATED'
  120 PRINT 'NOW WE ARE GOING TO SAVE 
     THE DATA! PRINT 'WHEN YOU ARE R 
     EADY START THE RECORDER IN RECOR 
     D MORE! : INPUT 'AND HIT RETURN'
     ,A$
  130 CALL -936: PRINT 'NOW WRITING DA 
     TA 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 DA 
     TA FROM TAPE'
  150 FOR I=1 TO 20: X(I)=: NEXT I
  160 PRINT 'NOW START TAPE RECORDER' 
     : INPUT 'AND THEN HIT RETURN'
     ,A$
  165 PRINT 'A ', A
  170 GOSUB 30
  180 PRINT 'AL 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 Ø of memory from $92 (2) to $14 (20). $00 (00) is used to store the relative period (P) between toggling of the speaker and $01 (01) is used as the memory location for the value of relative duration (Ø). Both P and D can range in value from $00 (0) to $FF (255). After the values for frequency and duration are placed into memory a CALL2 statement from BASIC will activate this routine. The speaker is toggled with the machine language statement residing at $02 and then a
delay in time equal to the value in $\text{ØØ}$ occurs. This process is repeated until the tone has lasted a relative period of time equal to the duration (value in $\text{Øl}$) and then this program is exited (statement $\text{l4}$).

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 $\text{ØØ}$ and $\text{Øl}$ and the CALL2 command activates the machine language program that toggles the speaker. After the tone has ended control is returned to the main program.

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

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

CONCLUSION

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

FIGURE 2. BASIC "POKES"

FIGURE 3. GOSUB
High-Resolution Operating Subroutines

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

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

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

In the following subroutine descriptions, all addresses given will be in decimal. The hexadecimal substitutes will be preceded by a dollar sign ($). All entry points given are for the cassette tape subroutines, which load into addresses C000 to FFF (hex). Equivalent addresses for the ROM subroutines will be in italic type face.
**High-Resolution Operating Subroutines**

**INIT** Initiates High-Resolution Graphics mode.
From BASIC: CALL 3072 (or CALL -12288)
From machine language: JSR $C00 (or JSR $D000)

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

**CLEAR** Clears the screen.
From BASIC: CALL 3886 (or CALL -12274)
From machine language: JSR SCOE (or JSR $L000E)

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

**PLOT** Plots a point on the screen.
From BASIC: CALL 3780 (or CALL -21589)
From machine language: JSR $C7C (or JSR $L107C)

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

PLOT (continued)

top of screen) to 159 (bottom of screen) and is passed in
location 802 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 800 (X MOD 256) and 801
(X/256). Or, from machine language, between registers X (X LO)
and Y (X HI). The color of the point to be plotted must be set
in location 812 ($32C). Four colors are possible: 0 is BLACK,
85 ($55) is GREEN, 170 ($AA) is VIOLET, and 255 ($FF) is WHITE.

POSN Positions a point on the screen.

From BASIC: CALL 3761 (or CALL -11599)

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

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

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

LINE  Draws a line on the screen.
   From BASIC: CALL 3786 (or CALL -11574)
   From machine language: JSR $C95 (or JSR $0995)

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

SHAPE  Draws a predefined shape on the screen.
   From BASIC: CALL 3805 (or CALL -11555)
   From machine language: JSR $DBC (or JSR $D1BC)

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

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

1 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 866 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 $90^\circ$ clockwise, etc. The rotation factor is passed in location 867 from BASIC of in the A-register from machine language.

The table of vectors which defines the shape to be drawn is a series of bytes stored in memory. Each byte is divided into three sections, and each section specifies whether or not to plot a point and also a direction to move (up, down, left, or right). The SHAPE subroutine steps through the vector table byte by byte, and then through each byte section by section. When it reaches a $00$ byte, it is finished.

The three sections are arranged in a byte like this:

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

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

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

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

Suppose we want to draw a shape like this:

First, draw it on graph paper, one dot per square. Then decide where to start drawing the shape. Let's start this one in the center.

Next, we must draw a path through each point in the shape, using only $90^\circ$ angles on the turns.

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

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

Now draw a table like the one in Figure 1. For each vector in the line, figure the bit code and place it in the next available section in the table. If it will not fit or is a $\emptyset$ at the end of a byte, then skip that section and go on to the next. When you have finished
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: \[ \begin{align*} \downarrow \downarrow & \leftarrow \uparrow \uparrow \uparrow \uparrow \rightarrow \rightarrow \rightarrow \rightarrow \downarrow \downarrow \downarrow \downarrow \end{align*} \]

Figure 1:

| C | B | A | C || A | \text{START HERE} |
|---|---|---|---|---|------------------|
| \emptyset | \emptyset | \emptyset | \emptyset | \emptyset | \emptyset |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 | 1 |
| 5 | 0 | 0 | 1 | 0 | 1 |
| 6 | 0 | 0 | 1 | 0 | 1 |
| 7 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 |

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

Figure 2:

<table>
<thead>
<tr>
<th>C</th>
<th>E</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \begin{array}{c|c|c|c|c|c|c|c|c} \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{array} = \begin{array}{c} 1 \text{Z} \end{array} \]

Hex-decimal Codes:

\[ \begin{array}{c|c|c} \emptyset & \emptyset & \emptyset \end{array} \rightarrow \begin{array}{c} 0 \text{O} \end{array} \]

\[ \begin{array}{c|c} \emptyset & \emptyset \end{array} \rightarrow \begin{array}{c} 1 \text{I} \end{array} \]

\[ \begin{array}{c} \emptyset \end{array} \rightarrow \begin{array}{c} 2 \text{J} \end{array} \]

\[ \begin{array}{c} \emptyset \end{array} \rightarrow \begin{array}{c} 3 \text{K} \end{array} \]

\[ \begin{array}{c} \emptyset \end{array} \rightarrow \begin{array}{c} 4 \text{L} \end{array} \]

\[ \begin{array}{c} \emptyset \end{array} \rightarrow \begin{array}{c} 5 \text{M} \end{array} \]

\[ \begin{array}{c} \emptyset \end{array} \rightarrow \begin{array}{c} 6 \text{N} \end{array} \]

\[ \begin{array}{c} \emptyset \end{array} \rightarrow \begin{array}{c} 7 \text{O} \end{array} \]

\[ \begin{array}{c} \emptyset \end{array} \rightarrow \begin{array}{c} 8 \text{P} \end{array} \]

\[ \begin{array}{c} \emptyset \end{array} \rightarrow \begin{array}{c} 9 \text{Q} \end{array} \]

\[ \begin{array}{c} \emptyset \end{array} \rightarrow \begin{array}{c} \text{A} \text{R} \end{array} \]

\[ \begin{array}{c} \emptyset \end{array} \rightarrow \begin{array}{c} \text{B} \text{S} \end{array} \]

\[ \begin{array}{c} \emptyset \end{array} \rightarrow \begin{array}{c} \text{C} \text{T} \end{array} \]

\[ \begin{array}{c} \emptyset \end{array} \rightarrow \begin{array}{c} \text{D} \text{U} \end{array} \]

\[ \begin{array}{c} \emptyset \end{array} \rightarrow \begin{array}{c} \text{E} \text{V} \end{array} \]

\[ \begin{array}{c} \emptyset \end{array} \rightarrow \begin{array}{c} \text{F} \text{W} \end{array} \]

\( \emptyset \emptyset \text{denotes end of vector table.} \)
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 4Ø high by 4Ø 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
130 COLOR=J+3/(I+3)+I*X/12
135 PLOT I,K: PLOT K,I: PLOT 40 -I,40-K
140 NEXT J,I
145 NEXT W: GOTO 105
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

5 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 12,204; POKE 13,4
16 POKE 14,190; POKE 15,24; POKE 16,248; POKE 17,5; POKE 18,
198; POKE 19,11; POKE 20,76;
POKE 21,2; POKE 22,8; POKE 23,36
15 DIM BK[40]; TEXT: CALL -936
: GOTO 90
20 CALL -936: GOTO 90
25 CALL LENS() FOR Z=1 TO 8: GOSUB 85: PRINT BK[Z,2];: NEXT Z:
GOSUB 78: RETURN
30 Bd=***********************
************: RETURN
35 Bd="COLOR SKETCH": RETURN
40 Bd=“COPYRIGHT APPLE COMPUTER 197
7: RETURN
45 Bd="THIS PROGRAM ALLOWS YOU TO "
: RETURN
50 Bd="SKECH COLORED FIGURES IN"
: RETURN
55 Bd="LOW RESOLUTION GRAPhS WITH
PADDELE": RETURN
60 IX=261:TON=28; GOSUB 85: RETURN
65 IX=10:TON=10; GOSUB 85: RETURN
70 IX=8:TON=50; GOSUB 85:IX=30
:TON=60: GOSUB 85: RETURN
75 IX=8:TON=258: GOSUB 85:IX=9
:TON=258: GOSUB 85: RETURN
85 IX=8:TON=258: GOSUB 85:IX=9
:TON=258: GOSUB 85: RETURN
95 POKE 1,TON MOD 256: POKE 24
,TON/256+1: POKE 8,IX,: CALL 3;
RETURN
90 GOSUB 38: GOSUB 25: PRINT :
TAB 19: GOSUB 38: GOSUB 25
: PRINT : GOSUB 38: GOSUB 25
: PRINT : TAB 5: GOSUB 49: GOSUB 25
51 PRINT : GOSUB 38: GOSUB 25
25
95 PRINT : GOSUB 78: GOSUB 45:
GOSUB 25: PRINT : GOSUB 50
: GOSUB 25: PRINT : GOSUB 55
: GOSUB 83: PRINT
100 PRINT : PRINT : GOSUB 78: INPUT
"WHEN READY HIT RETURN",Bd
105 OR
110 Bd="ABCDERFGHIJKLMNOP": CALL
-936
115 FOR Z=8 TO 15: COLOR=2: PLOT
Z+2+4.5: VTAB Z+11: GOSUB 78
+ TAB Z+2+5: PRINT BK[Z,2]: NEXT Z:
TAB 1
120 VTAB 32:04="TYPE A LETTER TO CH
CHANGE COLOR": GOSUB 25: PRINT
:BD="TYPE SPACE BAR TO STOP PLOT
.: GOSUB 25: PRINT
125 Y=总决赛(1):=GOSUB 25:B=PEEK (C
+0)+(Z+4): VTAB 04: TAB 1: PRINT
"CURSOR POSITION": Y="X: X=" Y="X:"
Y="X:"
130 IF PEEX (-16384) THEN 145
1 IF X1+X AND Y1+Y THEN 150
1: COLOR=C2: PLOT XI,Y1: IF NOT FLAG THEN 170: COLOR=C1:
: PLOT X,Y
135 C2=50: X1+X3=15: IF C2=
160 GOTO 125
135 IF PEEX (-16384) THEN 170
1:FLAG=0: POKE -16384,81 POKE
34,20: COLOR=9: MLINE 8,3: AT
30: CALL -936
150 PRINT :BD="CONTINUE OR STOP"
: VTAB 24: GOSUB 25: INPUT
"C/S": BD: IF BD(1,1)="C"
THEN 110: PRINT "END": END
155 FLAG=1:C= PEEX (-16384)+193
: POKE -16384,8: GOTO 125

58
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

0 REM GAME OF MASTERMIND  8-85-77
02 (APPLE COMPUTER)
10 REM RX(1,3,0,0,0),RX(0,3,0,0,0),RX(0,0,3,0,0)
12 RX(1,0,3,0,0),RX(0,1,3,0,0),RX(0,0,3,0,0)
14 RX(1,0,0,3,0),RX(0,1,0,3,0),RX(0,0,0,3,0)
16 RX(0,0,1,3,0),RX(0,0,0,3,0)
20 TEXT 1 CALL -566: PRINT " "

WELCOME

TO THE GAME OF MASTERMIND!

YOUR OBJECT IS TO GUESS 5 COLOR
5 CODES

30 PRINT "WILL MAKE UP A组合的 NUMBER OF GUESSES. THEN
32 USE EIGHT DIFFERENT COLORS TO

34 CODE FROM,"

40 PRINT " 

48 PRINT "MORE THAN 7 GUESSES---EXC

ELEGANT"; PRINT " 7 TO 9 GUESSES
"; PRINT "TO 14 GUESSES---AVG"
50 PRINT "MORE THAN 14 GUESSES---POOR"

60 CALL -304: TAB 7: PRINT
"HIT ANY KEY TO BEGIN PLAY"

100 CALL -304: IF PEEK <14608) THEN 100
110 PRINT "1 بو"; PRINT ""; PRINT ""; PRINT ""; PRINT ""; PRINT ""; PRINT ""
110 TRY: PRINT " TRY " LETTER
120 KEYS FOR COLOR CHANGE:1 PRINT
130 KEYS FOR REMOVE AND ADD
140 KEY: PRINT " HIT RETURN TO ADD
150 NEXT GUESS #";
PROGRAM DESCRIPTION
This program plots three Biorhythm functions: Physical (P), Emotional (E), and Mental (M) or intellectual. All three functions are plotted in the color graphics display mode.

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

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

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

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

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

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

5 POKE 2,173: POKE 3,680: POKE 4,193: POKE 5,169: POKE 6,8
: POKE 7,32: POKE 8,158: POKE 9,225: POKE 10,153: POKE 11
: POKE 12,280: POKE 13,4
10 POKE 14,199: POKE 15,24: POKE 16,249: POKE 17,5: POKE 18,
15 GOTO 25
20 TI=0: GOSUB 30: RETURN
25 PRINT "***************

***************": RETURN
30 XX=8:ST0M=20: GOSUB 45: RETURN
35 XX=8:ST0M=20: GOSUB 45: RETURN
40 XX=9:ST0M=22: GOSUB 45:XX=9
:ST0M=22: GOSUB 45: RETURN
45 POKE 1,7:POKE 256:POKE 24
: POKE 256+1:POKE 3,40: CALL 21: RETURN
50 N=19:POKE 116):M=1000K
: C1(1):K=1000:K1=100
: C1(1):C1(1):1500:11>1000
55 A=K:R=0:K0=1:K1=1:K2=1:R=0
: C1(1):C1(1):1500:11>1000
: F9:STO H=CALL -555
: POKE 94,26:GOSUB 29:GOSUB
: GOSUB 28:PRINT:TAB 18
: PRINT "APPLE II BIORHYTHM "<K>
: TAB 18: PRINT
55 GOSUB 29: TAB 18: PRINT "COPYRIGHT
: I 1977 "APPLE COMPUTER INC."
: POKE 14,20:VTAB 24
100 GOSUB 66: INPUT "NAME: ":A;
: VTAB 21: PRINT "AI:": VTAB 21
: PRINT "POCRIST": GOSUB 75
: VTAB 22: TAB 1: PRINT "BIRTH
: DATE:":A":":",":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":":"
DRAGON MAZE PROGRAM

PROGRAM DESCRIPTION
DRAGON MAZE is a game that will test your skill and memory. A maze is constructed on the video screen. You watch carefully as it is completed. After it is finished the maze is hidden as if the lights were turned out. The object of the game is to get out of the maze before the dragon eats you. A reddish-brown square indicates your position and a purple square represents the dragon's.* You move by hitting a letter on the keyboard; U for up, D for down, R for right, and L for left. As you advance so does the dragon. The scent of humans drives the dragon crazy; when he is enraged he breaks through walls to get at you. DRAGON MAZE is not a game for the weak at heart. Try it if you dare to attempt out-smarting the dragon.

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

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

10 PRINT "<NAME> TO THE DRAGON'S MAZE"
20 PRINT "WELCOME TO THE DRAGON'S MAZE"
30 PRINT "YOU MAY WATCH WHILE I BUILD A MAZE."
40 PRINT "BUT WHEN IT'S COMPLETE, I'LL ERASE IT."
50 PRINT "YOU WILL SEE THE WALLS AS YOU BUMP INTO THEM."
60 PRINT "TO MOVE, YOU Hit 'A' FOR RIGHT, 'L' FOR LEFT, 'U' FOR UP, AND 'D' FOR DOWN. DO NOT HIT RETURN."
70 PRINT "THE OBJECT IS FOR YOU (THE GREEN DOT)."
80 PRINT "GET TO THE DOOR ON THE RIGHT SIDE."
90 PRINT "BEFORE THE DRAGON (THE RED DOT) EATS YOU."
100 PRINT "YOU."
110 PRINT "BEWARE! SOMETIMES THE DRAGON CLIMBS OVER A WALL."
120 PRINT "BUT MOST OF THE TIME, HE CAN'T GO OVER."
130 PRINT "WHO HAS TO GO AROUND."
140 PRINT "YOU CAN OBTAIN TELL WHERE A WALL."
150 PRINT "IS, EVEN BEFORE YOU CAN SEE IT, BY"
160 PRINT "THE FACT THAT THE DRAGON CAN'T GET THROUGH IT."
170 PRINT "THE DRAGON CAN'T GET THROUGH IT."
180 PRINT "THE DRAGON CAN'T GET THROUGH IT."
190 PRINT "THE DRAGON CAN'T GET THROUGH IT."
200 PRINT "THE DRAGON CAN'T GET THROUGH IT."
210 PRINT "THE DRAGON CAN'T GET THROUGH IT."
220 PRINT "THE DRAGON CAN'T GET THROUGH IT."
230 PRINT "THE DRAGON CAN'T GET THROUGH IT."
240 PRINT "THE DRAGON CAN'T GET THROUGH IT."
250 PRINT "THE DRAGON CAN'T GET THROUGH IT."
260 PRINT "THE DRAGON CAN'T GET THROUGH IT."
270 PRINT "THE DRAGON CAN'T GET THROUGH IT."
280 PRINT "THE DRAGON CAN'T GET THROUGH IT."
290 PRINT "THE DRAGON CAN'T GET THROUGH IT."
300 PRINT "THE DRAGON CAN'T GET THROUGH IT."
310 PRINT "THE DRAGON CAN'T GET THROUGH IT."
320 PRINT "THE DRAGON CAN'T GET THROUGH IT."
330 PRINT "THE DRAGON CAN'T GET THROUGH IT."
340 PRINT "THE DRAGON CAN'T GET THROUGH IT."
350 PRINT "THE DRAGON CAN'T GET THROUGH IT."
360 PRINT "THE DRAGON CAN'T GET THROUGH IT."
370 PRINT "THE DRAGON CAN'T GET THROUGH IT."
380 PRINT "THE DRAGON CAN'T GET THROUGH IT."
390 PRINT "THE DRAGON CAN'T GET THROUGH IT."
400 PRINT "THE DRAGON CAN'T GET THROUGH IT."
410 PRINT "THE DRAGON CAN'T GET THROUGH IT."
420 PRINT "THE DRAGON CAN'T GET THROUGH IT."
430 PRINT "THE DRAGON CAN'T GET THROUGH IT."
440 PRINT "THE DRAGON CAN'T GET THROUGH IT."
450 PRINT "THE DRAGON CAN'T GET THROUGH IT."
460 PRINT "THE DRAGON CAN'T GET THROUGH IT."
470 PRINT "THE DRAGON CAN'T GET THROUGH IT."
480 PRINT "THE DRAGON CAN'T GET THROUGH IT."
490 PRINT "THE DRAGON CAN'T GET THROUGH IT."
500 PRINT "THE DRAGON CAN'T GET THROUGH IT."
510 PRINT "THE DRAGON CAN'T GET THROUGH IT."
520 PRINT "THE DRAGON CAN'T GET THROUGH IT."
530 PRINT "THE DRAGON CAN'T GET THROUGH IT."
540 PRINT "THE DRAGON CAN'T GET THROUGH IT."
550 PRINT "THE DRAGON CAN'T GET THROUGH IT."
560 PRINT "THE DRAGON CAN'T GET THROUGH IT."
570 PRINT "THE DRAGON CAN'T GET THROUGH IT."
580 PRINT "THE DRAGON CAN'T GET THROUGH IT."
590 PRINT "THE DRAGON CAN'T GET THROUGH IT."
600 PRINT "THE DRAGON CAN'T GET THROUGH IT."
610 PRINT "THE DRAGON CAN'T GET THROUGH IT."
620 PRINT "THE DRAGON CAN'T GET THROUGH IT."
630 PRINT "THE DRAGON CAN'T GET THROUGH IT."
640 PRINT "THE DRAGON CAN'T GET THROUGH IT."
650 PRINT "THE DRAGON CAN'T GET THROUGH IT."
660 PRINT "THE DRAGON CAN'T GET THROUGH IT."
670 PRINT "THE DRAGON CAN'T GET THROUGH IT."
680 PRINT "THE DRAGON CAN'T GET THROUGH IT."
690 PRINT "THE DRAGON CAN'T GET THROUGH IT."
700 PRINT "THE DRAGON CAN'T GET THROUGH IT."
710 PRINT "THE DRAGON CAN'T GET THROUGH IT."
720 PRINT "THE DRAGON CAN'T GET THROUGH IT."
730 PRINT "THE DRAGON CAN'T GET THROUGH IT."
740 PRINT "THE DRAGON CAN'T GET THROUGH IT."
750 PRINT "THE DRAGON CAN'T GET THROUGH IT."
760 PRINT "THE DRAGON CAN'T GET THROUGH IT."
770 PRINT "THE DRAGON CAN'T GET THROUGH IT."
780 PRINT "THE DRAGON CAN'T GET THROUGH IT."
790 PRINT "THE DRAGON CAN'T GET THROUGH IT."
800 PRINT "THE DRAGON CAN'T GET THROUGH IT."
810 PRINT "THE DRAGON CAN'T GET THROUGH IT."
820 PRINT "THE DRAGON CAN'T GET THROUGH IT."
830 PRINT "THE DRAGON CAN'T GET THROUGH IT."
840 PRINT "THE DRAGON CAN'T GET THROUGH IT."
850 PRINT "THE DRAGON CAN'T GET THROUGH IT."
860 PRINT "THE DRAGON CAN'T GET THROUGH IT."
870 PRINT "THE DRAGON CAN'T GET THROUGH IT."
880 PRINT "THE DRAGON CAN'T GET THROUGH IT."
890 PRINT "THE DRAGON CAN'T GET THROUGH IT."
900 PRINT "THE DRAGON CAN'T GET THROUGH IT."
910 PRINT "THE DRAGON CAN'T GET THROUGH IT."
920 PRINT "THE DRAGON CAN'T GET THROUGH IT."
930 PRINT "THE DRAGON CAN'T GET THROUGH IT."
940 PRINT "THE DRAGON CAN'T GET THROUGH IT."
950 PRINT "THE DRAGON CAN'T GET THROUGH IT."
960 PRINT "THE DRAGON CAN'T GET THROUGH IT."
970 PRINT "THE DRAGON CAN'T GET THROUGH IT."
980 PRINT "THE DRAGON CAN'T GET THROUGH IT."
990 PRINT "THE DRAGON CAN'T GET THROUGH IT."
1000 PRINT "THE DRAGON CAN'T GET THROUGH IT."
DRAGON MAZE cont.

12250 HR=36
12300 BY= RMSD (15)+1
12400 COLOR=6: VLIN 80+W-2, 3+6W-1

12500 SX=13: SY=W
12600 G0=3+6W-2: SY=W-2
12700 RX=1

1500 K = PEEK (-15364); IF K<128 THEN

15500 DX=0: DY=0

15600 IF SX=W AND SY=W THEN 2500
15700 IF K = ASC("R") THEN 2000
15800 IF K = ASC("L") THEN 2000
15900 IF K = ASC("U") THEN 3000
16000 IF K = ASC("D") THEN 3000
16100 GOSUB 9800: GOTO 1500
16200 IF MX+13*(SY-1) MOD 10 THEN 4000

16300 FA=3+W-2: FY=3+W-1: FOR I=1 TO 3
16400 FA=FA+6+W: FY=FY+6+W

26400 FOR K=0 TO 7: FOR L=0 TO 7:

26900 PLOT FA,K,LY+1: NEXT L,K
27000 NEXT K

27100 NEXT I
27150 K=25: XL=16
27200 IF K=13 AND Y=W THEN 8000
27300 GOTO 1500

27500 DX=0: DY=0
27550 IF MX+13*(SY-1) MOD 10 THEN 4100

7000 IF X=13 THEN 7000: IF Y>SY THEN
7050
7100 IF X=13 THEN 7100: IF Y>SY THEN
7150

7200 IF X=13 THEN 7200: IF Y<SY+13: IF T(ISX=13*(SY-1)+9) THEN 7210: IF
7260 IF ISX+13*(SY-1)+9 MOD 10 THEN
7300
7310 IF ISX+13*(SY-1)+9 THEN 7350
7400 FOR I=1 TO 3; RX=RX+6+W: RX=RX+6+W
7500 FOR I=1 TO 1: FOR L=0 TO 1:

7600 FOR I=1 TO 1: FOR L=0 TO 1:

7700 NEXT 1
7745 RETURN

7800 IF SY=13 THEN 7800: IF T(ISX=13*(SY-1)+9) THEN 7800: IF
7850 IF ISX+13*(SY-1)+9 MOD 10 THEN 7800
7860 IF ISX=13*(SY-1)+9: GOSUB 5000
7900 IF ISX=13*(SY-1)+9: GOSUB 5000: GOSUB 5000: GOSUB 5000

7900 PRINT "YOU WIN!"
7910 GOSUB 5000: GOSUB 5000: GOSUB 5000: GOSUB 5000

7920 PRINT "SCORE=",s+3
7930 END
7110 DX=-1;DY=0: GOTO 7020
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 7020
8000 GOSUB 5000: GOSUB 5000: GOSUB
5000: GOSUB 5000: PRINT 'THE DRA
GON GOT YOU!'
1999 END
APPLE II FIRMWARE

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

Apple II contains a powerful machine level monitor for use by the advanced programmer. To enter the monitor either press RESET button on keyboard or CALL-151 (Hex FF65) from Basic. Apple II will respond with an "*" (asterisk) prompt character on the TV display. This action will not kill current BASIC program which may be re-entered by a 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><strong>Examine Memory</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adres</td>
<td>*C0F2</td>
<td>Examines (displays) single memory location of (adrs)</td>
</tr>
<tr>
<td>adres1.adres2</td>
<td>*1024.1048</td>
<td>Examines (displays) range of memory from (adres1) thru (adres2)</td>
</tr>
<tr>
<td>(return)</td>
<td>*(return)</td>
<td>Examines (displays) next 8 memory locations</td>
</tr>
<tr>
<td>.adres2</td>
<td>*.4096</td>
<td>Examines (displays) memory from current location through location (adres2)</td>
</tr>
<tr>
<td><strong>Change Memory</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adres:data data</td>
<td>*A256:EF 20 43</td>
<td>Deposits data into memory starting at location (adres)</td>
</tr>
<tr>
<td>:data data data</td>
<td>*:F0 A2 12</td>
<td>Deposits data into memory starting after (adres) last used for deposits</td>
</tr>
<tr>
<td><strong>Move Memory</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adres1&lt;adres2. adres3M</td>
<td>*100&lt;B010.B410M</td>
<td>Copy the data now in the memory range from (adres2) to (adres3) into memory locations starting at (adres1)</td>
</tr>
<tr>
<td><strong>Verify Memory</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adres1&lt;adres2 adres3V</td>
<td>*100&lt;B010.B410V</td>
<td>Verify that block of data in memory range from (adres2) to (adres3) exactly matches data block starting at memory location (adres1) 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>adrs1.adrs2R</td>
<td>*300.4FFR</td>
<td>Reads cassette data into specified memory (adrs) range. Record length must be same as memory range or an error will occur.</td>
</tr>
<tr>
<td>adrs1.adrs2W</td>
<td>*800.9FFW</td>
<td>Writes onto cassette data from specified memory (adrs) range.</td>
</tr>
<tr>
<td><strong>Display</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>*I</td>
<td>Set inverse video mode. (Black characters on white background)</td>
</tr>
<tr>
<td>M</td>
<td>*N</td>
<td>Set normal video mode. (White characters on black background)</td>
</tr>
<tr>
<td><strong>Dis-assembler</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrsL</td>
<td>*C800L</td>
<td>Decodes 20 instructions starting at memory (adrs) into 6502 assembly mnemonic code.</td>
</tr>
<tr>
<td>L</td>
<td>*L</td>
<td>Decodes next 20 instructions starting at current memory address.</td>
</tr>
<tr>
<td><strong>Mini-assembler</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Turn-on)</td>
<td>*F666G</td>
<td>Turns-on mini-assembler. Prompt character is now a &quot;!&quot; (exclamation point).</td>
</tr>
<tr>
<td>$(monitor: command)</td>
<td>$C800L</td>
<td>Executes any monitor command from mini-assembler then returns control to mini-assembler. Note that many monitor commands change current memory address reference so that it is good practice to retype desired address reference upon return to mini-assembler.</td>
</tr>
<tr>
<td>adrs:(6502 MNEMONIC instruction)</td>
<td>!C010:STA 23FF</td>
<td>Assembles a mnemonic 6502 instruction into machine codes. If error, machine will refuse instruction, sound bell, and reprint line with up arrow under error.</td>
</tr>
<tr>
<td>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>adress</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adressG</td>
<td>*3ØØG</td>
<td>Runs machine level program starting at memory (adress).</td>
</tr>
<tr>
<td>adressT</td>
<td>*8ØØT</td>
<td>Traces a program starting at memory location (adress) 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 (adress). 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>*EØ</td>
<td>Displays 65Ø2 status registers and opens them for modification (see Note 1)</td>
</tr>
<tr>
<td>(Control Y)</td>
<td>*YØ</td>
<td>Executes user specified machine language subroutine starting at memory location (3F8).</td>
</tr>
</tbody>
</table>

### Note 1:

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

Example:  

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

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

### Hexidecimal Arithmetic

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

Note 2:

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

Multiple Commands

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

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

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

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

<table>
<thead>
<tr>
<th>Hex</th>
<th>BASIC Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><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 34,11</td>
<td>Set window top to line specified by T1 in range of 0 to 23</td>
</tr>
<tr>
<td>0023</td>
<td>120 POKE 35,B1</td>
<td>Set window bottom to line specified by B1 in the range of 0 to 23. B1&gt;T1</td>
</tr>
<tr>
<td>0024</td>
<td>130 CH=PEEK(36)</td>
<td>Read/set cursor horizontal position in the range of 0 to 39. If using TAB, you must add &quot;1&quot; to cursor position read value; Ex. 140 and 150 perform identical function.</td>
</tr>
<tr>
<td></td>
<td>140 POKE 36,CH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>150 TAB(CH+1)</td>
<td></td>
</tr>
<tr>
<td>0025</td>
<td>160 CV=PEEK(37)</td>
<td>Similar to above. Read/set cursor vertical position in the range 0 to 23.</td>
</tr>
<tr>
<td></td>
<td>170 POKE 37,CV</td>
<td></td>
</tr>
<tr>
<td></td>
<td>180 VTAB(CV+1)</td>
<td></td>
</tr>
<tr>
<td>0032</td>
<td>190 POKE 50,127</td>
<td>Set inverse flag if 127 (Ex. 190)</td>
</tr>
<tr>
<td></td>
<td>200 POKE 50,255</td>
<td>Set normal flag if 255(Ex. 200)</td>
</tr>
<tr>
<td>FC58</td>
<td>210 CALL -936</td>
<td>(@E) Home cursor, clear screen</td>
</tr>
<tr>
<td>FC42</td>
<td>220 CALL -958</td>
<td>(FE) Clear from cursor to end of page</td>
</tr>
<tr>
<td>Hex</td>
<td>BASIC Example</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>FC9C</td>
<td>23Ø CALL -868</td>
<td>(E\textsubscript{E}) 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>CØ3Ø</td>
<td>36Ø X=PEEK(-16336)</td>
<td>Toggle speaker</td>
</tr>
<tr>
<td></td>
<td>365 POKE -16336,0</td>
<td></td>
</tr>
<tr>
<td>CØ00</td>
<td>37Ø X=PEEK(-16384)</td>
<td>Read keyboard; if X&gt;127 then key was pressed.</td>
</tr>
<tr>
<td>CØ1Ø</td>
<td>38Ø POKE -16368,0</td>
<td>Clear keyboard strobe - always after reading keyboard.</td>
</tr>
<tr>
<td>CØ61</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>CØ62</td>
<td>40Ø X=PEEK(-16286)</td>
<td>Read PDL(1) push button switch.</td>
</tr>
<tr>
<td>CØ63</td>
<td>41Ø X=PEEK(-16285)</td>
<td>Read PDL(2) push button switch.</td>
</tr>
<tr>
<td>CØ58</td>
<td>42Ø POKE -16296,0</td>
<td>Clear Game I/O AN0 output</td>
</tr>
<tr>
<td>CØ59</td>
<td>43Ø POKE -16295,0</td>
<td>Set Game I/O AN0 output</td>
</tr>
<tr>
<td>CØ5A</td>
<td>44Ø POKE -16294,0</td>
<td>Clear Game I/O AN1 output</td>
</tr>
<tr>
<td>CØ5B</td>
<td>45Ø POKE -16293,0</td>
<td>Set Game I/O AN1 output</td>
</tr>
<tr>
<td>CØ5C</td>
<td>46Ø POKE -16292,0</td>
<td>Clear Game I/O AN2 output</td>
</tr>
<tr>
<td>CØ5D</td>
<td>47Ø POKE -16291,0</td>
<td>Set Game I/O AN2 output</td>
</tr>
<tr>
<td>CØ5E</td>
<td>48Ø POKE -16290,0</td>
<td>Clear Game I/O AN3 output</td>
</tr>
<tr>
<td>CØ5F</td>
<td>49Ø POKE -16289,0</td>
<td>Set Game I/O AN3 output</td>
</tr>
</tbody>
</table>
TITLE "APPLE II SYSTEM MONITOR"
LOC0 EPZ $00
LOC1 EPZ $01
WNDLFT EPZ $20
WNDWDT TH EPZ $21
WNDDTP EPZ $22
WNDBTM EPZ $23
CH EPZ $24
CV EPZ $25
GBASL EPZ $26
GBASH EPZ $27
BASL EPZ $28
BASH EPZ $29
BAS2L EPZ $2A
BAS2H EPZ $2B
H2 EPZ $2C
LMNEM EPZ $2C
RTNL EPZ $2C
V2 EPZ $2D
RMNEM EPZ $2D
RTNH EPZ $2D
MASK EPZ $2E
CHSUM EPZ $2E
FORMAT EPZ $2E
LASTIN EPZ $2F
LENGTH EPZ $2F
SIGN EPZ $2F
COLOR EPZ $30
MODE EPZ $31
INVLG EPZ $32
PROMPT EPZ $33
YSAV EPZ $34
YSAV1 EPZ $35
CSWL EPZ $36
CSWH EPZ $37
KSWL EPZ $38
KSWH EPZ $39
PCL EPZ $3A
PCI EPZ $3B
XQT EPZ $3C
A1L EPZ $3C
A1H EPZ $3D
A2L EPZ $3E
A2H EPZ $3F
A3L EPZ $40
A3H EPZ $41
A4L EPZ $42
A4H EPZ $43
A5L EPZ $44
A5H EPZ $45
ORG $F800 ROM START ADDRESS
F800: 4A PLOT LSR 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
F80C: 26 2E RTMASK STA MASK
F80E: B1 26 PLOT1 LDA (GBASL),Y DATA
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 MASK
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: 20 0E F8 JSR PLOT1 PLOT NEXT SQUARE
F824: 90 F6 BCC HLINE1 ALWAYS TAKEN
F826: 48 VLINE JSR PLOT5 PLOT SQUARE
F828: 68 PLA
F82A: 85 27 STY V2 STORE AS BOTTOM COORD
F82C: A9 00 LDA #$00 NEXT LEFTMOST X-COORD
F82E: 84 2D CLRSC2 STY V2 STORE AS BOTTOM COORD
F830: 68 PLA
F832: A0 2F CLRSC3 LDA #$2F MAX Y, FULL SCRN CLR
F834: D0 02 BNE CLRSC2 ALWAYS TAKEN
F836: A0 27 CLETOP LDA #$27 MAX Y, TOP SCREEN CLR
F838: 84 2D CLRSC2 CPY V2 DONE?
F83A: A0 27 LDA #$27 RIGHTMOST X-COORD (COLUMN)
F83C: A9 00 CLRSC3 LDA #$50 TOP COORD FOR VLINE CALLS
F83E: 85 30 STA COLOR CLEAR COLOR (BLACK)
F840: 20 28 F8 JSR VLINE DRAW VLINE
F844: 88 DEY NEXT LEFTMOST X-COORD
F846: 10 F6 BPL CLRSC3 LOOP UNTIL DONE
F848: 60 RTS
F84A: 2F PHA FOR INPUT 000DEFGH
F84C: 4A LSR
F84E: 49 03 AND #$03
F850: 0E 09 ORA #$04 GENERATE GBASH=000000FG
F854: 69 7F ADC #$7F
F856: 85 26 GBASCALC 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=17*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 LSR (CARRY)
F873: 20 47 F8 JSR GBASCALC CALC BASE ADDRESS
F875: B1 26 LDA (GBASL),Y GET BYTE
F877: 28 PLP RESTORE LSR FROM CARRY
F879: 90 04 SCRN2 ECC RTMSK2 IF EVEN, USE LO H
F87B: 4A LSR A
F87C: 4A LSR A
F87D: 4A LSR A SHIFT HIGH HALF BYTE DOWN
F87E: 4A LSR A
F87F: 29 0F RTMSK2 AND #$5F MASK 4-BITS
F881: 60 RTS
F882: A6 3A INSDSL1 LDX PCL PRINT PCL,H
F884: A4 3B LDY PCH FOLLOWED BY A BLANK
F886: 20 96 FD JSR PRTYS2
F888: 20 48 F9 JSR PRTLNX FOLLOWED BY A BLANK
F88C: A1 3A STA COLOR (PCL,X) GET OP CODE
F88D: A8 INSDSL2 TAY EVEN/ODD TEST
F890: 90 09 BCC IEVEN
F892: 6A ROF BIT 1 TEST
F893: B0 10 RSC ERR XXXX111 INVALID OP
F895: C9 A2 CMP #$A2
F897: F0 0C BCC 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 SCRN2 R/L H-BYTE ON CARRY
F8A3: D0 04 BNE GETFMT TAX
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-#gt00101XXX
F8C3: 09 20 ORA #$20 2) XXXXXX11 INVALID OP
F8C5: 88 DEY 3) XXXX0100-#gt00111XXX
F8C6: D0 FA BNE MNNDX3 4) XXXX0100-#gt00111XXX
F8C8: C8 INY 5) XXXX0100-#gt00111XXX
F8CA: 98 MNNDX3 DEY
F8CC: D0 F2 BNE MNNDX1
F8CD: 60 RTS
F8DD: FF FF FF DBF SPP,SFF,SPF
F8E0: 82 F8 INSTDSP JSR INSDSL1 GEN PFT, LEN BYTES
F8E2: 48 PHA SAVE MNEMONIC TABLE INDEX
F8E4: B1 3A PRINTOP LDA (PCL),Y
F8E6: 20 DA FD JSR PRBYTE
F8E9: A2 01 LDX #$01 PRINT 2 BLANKS
F8E9: 20 4A F9 PRINTL JSR PBBL2
F8ED: C4 2F CPY LENGTH PRINT INST (1-3 BYTES)
F8EE: C8 INY IN A 12 CHR FIELD
F8EF: 90 F1 BCC PRINTOP
F8F1: A2 03 LDX #$03 CHAR COUNT FOR MNEMONIC PRINT
F8F2: C0 04 CPY #$04
DIV: 20 A4 FB DIVPM JSR MD1 ABS VAL OF AC, AUX.
DIV: A0 10 DIV LDY #$10 INDEX FOR 16 BITS
DIV2: 06 50 ASL ACL
DIV: 26 51 ROL AC
DIV: 26 52 ROL XTNDL XTND/AUX
DIV: 26 53 ROL XTNDH TO AC.
DIV: 38 SEC
DIV: A5 52 LDA XTNDL
DIV1: E5 54 SBC AUXL MOD TO XTND.
DIV: AA: 20 A4 FB DIVPM JSR MD1 ABS VAL OF AC, AUX.
DIV: A0 10 DIV LDY #$10 INDEX FOR 16 BITS
DIV2: 06 50 ASL ACL
DIV: 26 51 ROL AC
DIV: 26 52 ROL XTNDL XTND/AUX
DIV: 26 53 ROL XTNDH TO AC.
DIV: 38 SEC
DIV: A5 52 LDA XTNDL
DIV: E5 54 SBC AUXL MOD TO XTND.
DIV: AA: 20 A4 FB DIVPM JSR MD1 ABS VAL OF AC, AUX.
DIV: A0 10 DIV LDY #$10 INDEX FOR 16 BITS
DIV2: 06 50 ASL ACL
DIV: 26 51 ROL AC
DIV: 26 52 ROL XTNDL XTND/AUX
DIV: 26 53 ROL XTNDH TO AC.
DIV: 38 SEC
DIV: A5 52 LDA XTNDL
DIV: E5 54 SBC AUXL MOD TO XTND.
FC1E: B0 0B              BCS   RTS4       IF TOP LINE THEN RETURN
FC20: A5 25              VTABZ LDA CV         GET CURSOR V-INDEX
FC24: 20 21               JSR   BASCALC    GENERATE BASE ADDR
FC27: 65 20              ADC MWEFT ADD WINDOW LEFT INDEX
FC29: 85 28              STA BASL TO BASL
FC2B: 60                 RTS RTS
FC2C: 49 C0     ESC1     EOR #$C0       ESC?
FC2E: F0 28              BEQ   HOME IF SO, DO HOME AND CLEAR
FC30: 69 FD              ADC #$FD       ESC-A OR B CHECK
FC32: 90 C0              BCC   ADVANCE A, ADVANCE
FC34: F0 DA              BEQ   BS           B, BACKSPACE
FC36: 69 FD              ADC #$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 P, RETURN
FC42: A4 24     CLREOP   LDY   CH         CURSOR H TO Y INDEX
FC44: A5 25              LDA   CV         CURSOR V TO A-REGISTER
FC46: 48                 CLEOP1 PHA SAVE CURRENT LINE ON STK
FC47: 20 24 FC           JSR   VTABZ CALC BASE ADDRESS
FC49: 20 24 FC           JSR   VTABZ CLEAR TO EOL, SET CARRY
FC4D: A0 00              LDY #$00 CLEAR FROM H INDEX=0 FOR REST
FC4F: 68                 PLA INCREMENT CURRENT LINE
FC50: 69 00              ADC #$00        (CARRY IS SET)
FC52: C5 23              CMP WNDBTM DONE TO BOTTOM OF WINDOW?
FC54: 90 F0              BCC CLEOP1 NO, KEEP CLEARING LINES
FC56: B0 CA              BCS VTAB Y=, TAB TO CURRENT LINE
FC58: A5 22 HOME LDA WNDTOP INIT CURSOR V
FC5A: 85 25              STA CV AND H-INDICES
FC5C: A0 00              LDY #$00
FC5E: 84 24              STY CH THEN CLEAR TO END OF PAGE
FC60: F0 E4              BEQ CLEOP1
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 INC CURSOR V(DOWN 1 LINE)
FC68: A5 25              LDA CV
FC6A: C5 23              CMP WNDBTM OFF SCREEN?
FC6C: 90 B6              BCC VTABZ NO, SET BASE ADDR
FC6E: C6 25              DEC CV DECK CURSOR V (BACK TO BOTTOM)
FC70: A5 22 SCROLL LDA WNDTOP START AT TOP OF SCRL WNDW
FC72: 48                 PHA
FC73: 20 24 FC           JSR   VTABZ GENERATE BASE ADR
FC75: A5 28              SCLRL LDA BASL COPY BASL,H
FC77: 85 2A              STA BASL TO BASL,H
FC79: A5 29              LDA BASL
FC7C: 85 28              STA BASLH
FC7E: A4 21              LDY MWDNTH INIT Y TO RIGHTMOST INDEX
FC80: 88                 DEY OF SCROLLING WINDOW
FC81: 68                 PLA
FC82: 69 01              ADC #$01 INCR LINE NUMBER
FC84: C5 23              CMP WNDBTM DONE?
FC86: B0 0D              BCS SCRL3 YES, FINISH
FC88: 48                 PHA
FC89: 20 24 FC           JSR VTABZ FORM BASL,H (BASE ADDR)
FC8C: B1 28              SCLRL LDA (BASL),Y MOVE A CHR UP ON LINE
FC8E: 91 2A              STA (BASL),Y
FC90: 88                 DEY NEXT CHAR OF LINE
FC92: 10 F9              BPL SCRL2
FC94: 30 E1              BMI SCRL1 NEXT LINE (ALWAYS TAKEN)
FC95: A0 00              SCRL3 LDY #$00 CLEAR BOTTOM LINE
FC97: 20 9E FC           JSR CLEOL2 GET BASE ADDR FOR BOTTOM LINE
FC99: B0 06              BCS VTAB CARRY IS SET
FC9B: A4 24     CLEOL LDY CH CURSOR H INDEX
FC9D: A9 A0     CLEOL2 STA (BASL),Y STORE BLANKS FROM 'HERE'
FC9F: A4 21              LDY MWDNTH TO END OF LINES (MWDNTH)
FCA0: 88                 DEY OF SCROLLING WINDOW
FCA1: 68                 PLA
FCA2: 69 01              ADC #$01 INCR LINE NUMBER
FCA4: C5 23              CMP WNDBTM DONE?
FCA6: B0 0D              BCS SCRL3 YES, FINISH
FCA8: 48                 PHA
FCA9: 20 24 FC           JSR VTABZ FORM BASL,H (BASE ADDR)
FCAB: B1 28              SCLRL LDA (BASL),Y MOVE A CHR UP ON LINE
FCAD: 91 2A              STA (BASL),Y
FCAE: 88                 DEY NEXT CHAR OF LINE
FCB0: 10 F9              BPL SCRL2
FCB2: 30 E1              BMI SCRL1 NEXT LINE (ALWAYS TAKEN)
FCB3: A0 00              SCRL3 LDY #$00 CLEAR BOTTOM LINE
FCB5: 20 9E FC           JSR CLEOL2 GET BASE ADDR FOR BOTTOM LINE
FCB7: B0 06              BCS VTAB CARRY IS SET
FCB9: A4 24     CLEOL LDY CH CURSOR H INDEX
FCBB: A9 A0     CLEOL2 STA (BASL),Y STORE BLANKS FROM 'HERE'
FCBD: A4 21              LDY MWDNTH TO END OF LINES (MWDNTH)
FD67:  20  8E  FD  GETLNZ  JSR  CROUT  OUTPUT CR
FD6A:  A5  33  GETLN  LDA  PROMPT
FD6C:  A2  01  LDX  #$01  INIT INPUT INDEX
FD71:  8A  BCKSPC  TXA  WILL BACKSPACE TO 0
FD72:  F0  F3  BEQ  GETLNZ
FD74:  CA  DEX
FD75:  20  35  FD  NXTCHAR  JSR  RDCHAR
FD78:  C9  95  CMP  #PICK  USE SCREEN CHAR
FD7A:  D0  02  BNE  CAPTST  FOR CTRL-U
FD7C:  B1  28  LDA  (BASI),Y

FD7E:  C9  E0  CAPTST  CMP  #$E0
FD80:  90  02  BCC  ADDIMP  CONVERT TO CAPS
FD82:  29  DF  AND  #$DF
FD84:  9D  02  ADDIMP  STA  IN,X  ADD TO INPUT BUF
FD87:  C9  BD  CMP  #$BD
FD89:  D0  B2  BNE  NOTCR  DON'T OUTPUT CR
FD8B:  20  8E  FD  GETLN  JSR  CROUT  OUTPUT CR
FD8E:  A9  8D  LDA  #$8D  PRINT CR,A1 IN HEX
FD92:  A4  3D  PRA1  LDY  A1H  PRINT CR,A1 IN HEX
FD94:  A6  3C  LDX  A1L
FD96:  20  8E  FD  PRYX2  JSR  CROUT  OUTPUT CR
FD99:  20  40  F9  JSR  PRNTYX
FD9C:  A0  00  LDY  #$00
FD9E:  A9  AD  LDA  #$AD  PRINT '-'

FDAA:  3D  E0  JMP  COUT  OUTPUT BLANK
FDAC:  B1  3C  LDA  A1L  BLANK TO MON
FDAD:  85  3F  STA  A2H
FDAE:  A5  3C  MODSCHK  LDA  A1L

FD03:  86  PRBYTE  PHA  PRINT BYTE AS 2 HEX
FD05:  4A  LSR  A  DETERMINE MON MODE
FD07:  4A  LSR  A
FD09:  4A  LSR  A
FD0B:  4A  LSR  A
FD0D:  4A  LSR  A
FD0F:  4A  LSR  A
FD11:  69  06  ADC  #$06  THEN RETURN
FDC2:  6C  36  00  COUT  JMP  (CSWL)  VECTOR TO USER OUTPUT ROUTINE
FDC6:  C9  00  COUT1  JMP  (CSWL)
FDC9:  85  31  STOR  STA  MODE  KEEP IN STORE MODE
FDD2:  25  32  AND  #$32  MASK WITH INVERSE FLAG
FDD4:  85  36  JMP  (CSWL)  VECTOR TO USER OUTPUT ROUTINE
FDD8:  C9  A0  COUT  JMP  (CSWL)
FDDA:  20  8E  FD  GETLN  JSR  CROUT  OUTPUT CR
FDDE:  20  8E  FD  GETLN  JSR  CROUT  OUTPUT CR
FDFF:  86
```
FEB6: 20 75 FE GO JSR A1PC ADR TO PC IF SPEC'D
FEB9: 20 3F FF JSR RESTORE RESTORE MTA REGS
FEBC: 6C 3A 00 JMP (PCL) GO TO USER SUBR
FEBF: 4C D7 AA JMP REGDSP TO REG DISPLAY
FEC2: 6C 3A 00 JMP (PCL) GO TO USER SUBR
FEC5: 4C D7 AA JMP REGDSP TO REG DISPLAY
FEC8: 20 ED FE JSR WRBYTE WRITE 10-SEC HEADER
FECB: 20 ED FE JSR WRBYTE WRITE 10-SEC HEADER
FECF: 20 ED FE JSR WRBYTE WRITE 10-SEC HEADER
FED2: A0 27 LDY #$27
FED4: A2 00 WR1 LDY #$00
FED6: A1 3C EOR (A1L,X)
FED8: 48 PHA
FEDA: A1 3C EOR (A1L,X)
FEDC: A9 40 WR1 LDY #$40
FEDF: A0 1D LDY #$1D
FEE1: A0 22 LDY #$22
FEE3: A0 24 RD2 LDY #$24
FEE5: 90 EE BCC WR1
FEE7: A0 3B LDY #$3B
FEE9: 20 ED FE JSR WRBYTE WRITE 10-SEC HEADER
FEEB: F0 4D BEQ BELL
FEED: A2 10 WRBYTE LDY #$10
FEEF: 0A WRBYTE2 ASL A
F700: A9 16 LDA #$16 DELAY 3.5 SECONDS
F702: 85 2E STA CHKSUM INIT CHKSUM=$FF
F704: A0 24 RD2 LDY #$24
F706: 20 EC FC JSR RDBYTE READ A BYTE
F709: A9 31 STA (A1L,X) STORE AT (A1)
F70B: 45 2E EOR CHKSUM
F70D: 28 PLP
F70F: 60 RTS
F711: A9 35 STA CHKSUM UPDATE RUNNING CHKSUM
F713: 20 BA PC JSR NXTA1 INC A1, COMPARE TO A2
F715: A0 35 RD2 LDY #$35 COMPENSATE 0/1 INDEX
F717: A0 3B LDY #$3B INDEX FOR 0/1 TEST
F719: A9 87 BELL LDA #$87 OUTPUT BELL AND RETURN
F71B: 4C ED FD JMP COUT
F71D: A5 48 RESTORE LDA STATUS RESTORE 6502 REG CONTENTS
F721: 48 PHA
F723: A5 48 RESTORE LDA STATUS RESTORE 6502 REG CONTENTS
F725: 8A 46 RESTR1 LDX XREG
F727: 84 47 LDY YREG
F729: 28 PLP
F72B: A5 48 RESTORE LDA STATUS RESTORE 6502 REG CONTENTS
F72D: A5 48 RESTORE LDA STATUS RESTORE 6502 REG CONTENTS
F72F: 20 FF FB JSR SETVID AS I/O DEV'S
F731: 20 2F FB JSR INIT AND INIT KEI/SCREEN
F733: A9 99 JSR SETVID AS I/O DEV'S
F735: A9 AA LDA #$AA `'` PROMPT FOR MON
F737: A9 AA LDA #$AA `'` PROMPT FOR MON
F739: 85 33 STA PROMPT
F73B: 20 67 FD JSR GETMN READ A LINE
```
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 CHRSRC  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 CHRSRC
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 #$03
FF8C: 0A ASL A
FF8D: 0A ASL A
FF8E: 0A ASL A
FF91: 26 3E ROL A2L
FF93: 26 3F ROL A2H
FF95: CA DEX  LEAVE X=$FF IF DIG
FF96: 10 F8 BPL NXTBIT
FF98: A5 31 NXTBS LDA MODE
FF9A: D0 06 BNE NXTBS2  IF MODE IS ZERO
FF9C: B5 3F LDA A2H,X  THEN COPY A2 TO
FF9E: 95 3D STA A1H,X  A1 AND A3
FFA0: 95 41 STA A3H,X
FFA2: E8 NXTBS INX
FFA3: F0 P3 BEQ NXTBS
FFA5: D0 06 LDA N NXTCHR
FFA7: A2 00  GETNUM LDX #$00  CLEAR A2
FFA9: 86 3E STX A2L
FFAB: 86 3F STX A2H
FFAD: B9 00 2S NXTCHR LDA IN,Y  GET CHAR
FFB0: C8 INY
FFB1: 49 B0 EOR #$B0
FFB3: C9 0A CMP #$0A
FFB5: 90 D3 BCC DIG  IF HEX DIG, THEN
FFB7: 69 88 ADC #$88
FFB9: C9 FA CMP #$FA
FFBA: B0 CD BCS DIG
FFBD: 60 RTS
FFBE: A9 FE TOSUB LDA #0/256  PUSH HIGH-ORDER
FFC0: 48 PHA  SUBR ADR ON STK
FFC1: B9 E3 FF LDA SUBLT,Y  PUSH LOW-ORDER
FFC4: 48 PHA  SUBR ADR ON STK
FFC5: A5 31 LDA MODE
FFC7: A0 00 ZMODE LDY #$00  CLR MODE, OLD MODE
FFC9: 84 31 STY MODE  TO A-REG
FFCA: 60 RTS  GO TO SUBR VIA RTS
FFCC: BC CHRTBL DFB $BC  F("CTRL-C")
FFCD: B2 DFB $B2  F("CTRL-Y")
FFCE: BE DFB $BE  F("CTRL-B")
FFCF: ED DFB $ED  F("T")
FFD0: EF DFB $EF  F("V")
FFD1: C4 DFB $C4  F("CTRL-K")
FFD2: EC DFB $EC  F("S")
FFD3: A9 DFB $A9  F("CTRL-P")
FFD4: BA DFB $BA  F("CTRL-B")
FFD5: A6 DFB $A6  F("-")
FFD6: A4 DFB $A4  F("-")
FFD7: 06 DFB $06  F("M")  {F-EX-OR #$B0+$89}
FFD8: 95 DFB $95  F("1t")
FFD9: 07 DFB $07  F("N")
FFDA: 02 DFB $02  F("I")
FFDB: 05 DFB $05  F("L")
FFDC: F0 DFB $F0  F("W")
FFDD: 00 DFB $00  F("Q")
FFDE: EB DFB $EB  F("R")
FFDF: 93 DFB $93  F(":")
FFE0: A7 DFB $A7  F(".")
FFE1: C6 DFB $C6  F("CR")
FFE2: 99 DFB $99  F("BLANK")
FFE3: B2 SUBTBL DFB BASCONT-1
FFE4: C9 DFB USR-1
FFE5: BE DFB REGZ-1
FFE6: C1 DFB TRAC-1
FFE7: 35 DFB VPY-1
FFE8: 8C DFB INPRT-1
FFE9: C3 DFB STEPZ-1
FFE9: 96 DFB OUTPRT-1
FFE9: AP DFB XBASIC-1
FFE9: 17 DFB SETMODE-1
FFED: 17 DFB SETMODE-1
FFE8: 2B DFB MOVE-1
FFE9: 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
FFFFD: FF    DFB  RESET/256
FFFFE: 86    DFB  IRQ      IRQ VECTOR
FFFFF: FA    DFB  IRQ/256

XQTNZ    EQU   $3C
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 $F9B5
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
CHARBL 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     LBY A2H
F507: A6 3E     LDS A2L DOUBLE DECREMENT
F509: D0 01     BNE REL2
F50B: 88      DEY
F50C: CA       REL2 DBX
F50D: 8A      TXA
F50E: 18      CLC
F50F: E5 3A    SBC PCL FORM ADDR-PC-2
F511: 85 3E    STA A2L
F513: 10 01    BPL REL3
F515: C8      INY
F516: 98       REL3 TYA
F5CC: 26 42  ROL  A4L
F5CE: 26 43  ROL  A4K
F5D0: CA  DEX
F5D1: 10 F8  BPL  NXTM2
F5D3: C6 3D  DEC  A1H  DONE WITH 3 CHAR?
F5D5: F0 F4  BEQ  NXTM2  YES, BUT DO 1 MORE SHIFT
F5D7: 10 E4  BPL  NXTMN  NO
F5D9: A2 05  FORM1  LDX  $85  5 CHARs IN ADDR MODE
F5DB: 20 34 F6  FORM2  JSR  GETNSP  GET FIRST CHAR OF ADDR
F5DE: 84 34  STY  YSAV
F5E0: DD B4 F9  CMP  CHAR1,X  FIRST CHAR MATCH PATTERN?
F5E3: D0 13  BNE  FORM3  NO
F5E5: 20 34 F6  JSR  GETNSP  YES, GET SECOND CHAR
F5E8: DD BA F9  CMP  CHAR2,X  MATCHES SECOND HALF?
F5EB: F0 0D  BEQ  FORM5  YES.
F5ED: BD BA F9  LDA  CHAR2,X  NO, IS SECOND HALF ZERO?
F5F0: F0 07  BEQ  FORM4  YES.
F5F2: C9 A4  CMP  #$A4  NO, SECOND HALF OPTIONAL?
F5F4: F0 03  BEQ  FORM4  YES.
F5F6: A4 34  LDY  YSAV
F5F8: 18  FORM3  CLC  CLEAR BIT-NO MATCH
F5FA: 88  FORM4  DEX  BACK UP 1 CHAR
F5FCA: 26 44  FORM5  ROL  PMT  FORM FORMAT BYTE
F5FC: E0 03  CPX  #$3  TIME TO CHECK FOR ADDR.
F5FD: BD 0D  BNE  FORM7  NO
F600: 20 A7 FF  JSR  GETNUM  YES
F603: A5 3F  LDA  A2H
F605: F0 01  BEQ  FORM6  HIGH-ORDER BYTE ZERO
F607: E8  INX  NO, INCR FOR 2-BYTE
F608: 86 35  STX  L  STORE LENGTH
F60A: A2 03  LDX  #$3  RELOAD FORMAT INDEX
F60C: 88  DEY  BACKUP A CHAR
F60D: 86 3D  STX  A1H  SAVE INDEX
F60F: CA  DEX  DONE WITH FORMAT CHECK?
F610: 10 C9  BPL  FORM2  NO.
F612: A5 44  LDA  PMT  YES, PUT LENGTH
F614: 0A  ASL  A  IN LOW BITS
F615: 0A  ASL  A
F616: 05 35  ORA  L
F618: C9 20  CMP  #$20
F61A: B0 06  BCS  FORM8  ADD "$" IF NONZERO LENGTH
F61C: A6 35  LDX  L  AND DON'T ALREADY HAVE IT
F61E: F0 02  BEQ  FORM8
F620: 09 80  ORA  #$80
F622: 85 44  FORM8  STA  PMT
F624: 84 34  STY  YSAV
F626: B9 00 02  LDA  IN,Y  GET NEXT NONBLANK
F629: C9 BB  CMP  #$BB  "" START OF COMMENT?
F62B: F0 04  BEQ  FORM9  YES
F62D: C9 8D  CMP  #$8D  CARRIAGE RETURN?
F62F: D0 80  BNE  ERR4  NO, ERR.
F631: 4C 5C F5  FORM9  JMP  TRYNEXT
F634: B9 00 02  GETNSP  LDA  IN,Y
F637: C8  INY
F638: C9 A0  CMP  #$A0  GET NEXT NON BLANK CHAR
F63A: F0 F8  BEQ  GETNSP
F63C: 60  RTS
F666: 4C 92 F5  MINIASM  JMP  RESETZ
**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

**ADD** CLC CLEAR CARRY

**LDX** #$2 INDEX FOR 3-BYTE ADD.

**LDA** M1,X

**ADC** M2,X ADD A BYTE OF MANT2 TO MANT1

**STA** M1,X

**DEX** INDEX TO NEXT MORE SIGNIFICANT BYTE.

**BPL** ADD1 LOOP UNTIL DONE.

**RTS**

**ASL** SIGN CLEAR LSB OF SIGN.

**JSR** ABSWAP ABS VAL OF M1, THEN SWAP WITH M2

**BIT** M1 MANT1 NEGATIVE?

**BPL** ABSWAP1 NO, SWAP WITH MANT2 AND RETURN.

**JSR** FCOMPL YES, COMPLEMENT IT.

**SEC** INCR SIGN, COMPLEMENTING LSB.

**JSR** ABSWAP1 SET CARRY FOR RETURN TO MUL/DIV.

**LDX** #$4 INDEX FOR 4 BYTE SWAP.

**LDA** X1-1,X

**LDY** X2-1,X

**STY** X1-1,X

**STA** X2-1,X

**DEX** ADVANCE INDEX TO NEXT BYTE

**BNE** SWAP1 LOOP UNTIL DONE.

**RTS**

**JSR** FCOMPL CMPL MANT1,CLEARS CARRY UNLESS 0

**JSR** ALGNSWP RIGHT SHIFT MANT1 OR SWAP WITH

**JSR** ADD ADD ALIGNED MANTISSAS.

**BVC** NORM NO OVERFLOW, NORMALIZE RESULT.

**BMI** RTS1 YES, RETURN WITH MANT1 NORMALIZED

**BMI** RTS1 DECUMENT EXP1.

**ASL** M1+2

**ROL** M1+1 SHIFT MANT1 (3 BYTES) LEFT.

**ROL** M1

**LDA** X1 EXP1 ZERO?

**BNE** NORM1 NO, CONTINUE NORMALIZING.

**RTS**

**JSR** FCOMPL CMP M1,CLEARS CARRY UNLESS 0

**JSR** ALGNSWP RIGHT SHIFT MANT1 OR SWAP WITH

**LDA** X2

**CMP** X1 COMPARE EXP1 WITH EXP2.

**BNE** SWALGN IF #,SWAP ADDENDS OR ALIGN MANTS.

**JSR** ADD ADD ALIGNED MANTISSAS.

**BVC** NORM NO OVERFLOW, NORMALIZE RESULT.

**BVS** RTLOG CV: SHIFT M1 RIGHT, CARRY INTO SIGN
F47B: 90 C4
F47C: ALGNSWP
F47D: SWAP
F47E: ELSE SHIFT RIGHT ARITH.
F47F: 0A
F480: ASL
F481: RIGHT ARITH SHIFT.
F482: 7F 75
F483: BBQ
F484: OVFL
F485: EXPI OUT OF RANGE.
F486: 76 PF
F487: RTLOG1
F488: LX#1 $1F
F489: INDEX FOR 6:BYTE RIGHT SHIFT.
F48A: 8E
F48B: INX
F48C: NEXT BYTE OF SHIFT.
F48D: 60
F48E: RTS
F48F: RETURN.
F490: 20 32 F4
F491: PMUL
F492: JSR
F493: MD1
F494: ABS VAL OF MANT1, MANT2
F495: 65 F8
F496: ADC
F497: X1
F498: ADD EXP1 TO EXP2 FOR PRODUCT EXP
F499: A2 FA
F49A: RTLOG1
F49B: LDX
F49C: #$FA
F49D: INDEX FOR 6:BYTE RIGHT SHIFT.
F49E: 76 FF
F49F: ROR1
F4A0: ROR  E+3,X
F4A1: E8
F4A2: INX
F4A3: NEXT BYTE OF SHIFT.
F4A4: F0 C5
F4A5: BEQ
F4A6: OVFL
F4A7: EXP1 OUT OF RANGE.
F4A8: 38
F4A9: FCOMPL
F4AA: SEC
F4AB: SET CARRY FOR SUBTRACT.
F4AC: A2 03
F4AD: LDX
F4AE: #$3
F4AF: INDEX FOR 3 BYTE SUBTRACT.
F4B0: A9 00
F4B1: COMPL1
F4B2: LDA
F4B3: #$0
F4B4: CLEAR A.
F4B5: F5 F8
F4B6: SBC X1,X
F4B7: SUBTRACT BYTE OF EXP1.
F4B8: 95 F8
F4B9: STA X1,X
F4BA: RESTORE IT.
F4BB: CA
F4BC: DEX
F4BD: NEXT MORE SIGNIFICANT BYTE.
F4BE: 46 F3
F4BF: MDEND
F4C0: LSR SIGN
F4C1: TEST SIGN LSB.
F4C2: 90 BF
F4C3: NORMX
F4C4: BCC NORM
F4C5: IF EVEN,NORMALIZE PROD,ELSE COMP
F4C6: 38
F4C7: FCOMPL
F4C8: SBC X1,X
F4C9: SUBTRACT BYTE OF EXP1.
F4CA: 95 F8
F4CB: STA X1,X
F4CC: RESTORE IT.
F4CD: 26 F3
F4CE: DIV3
F4CF: STA M2+3,X
F4D0: 95 F8
F4D1: STAX M1+2
F4D2: 86 FB
F4D3: DIV1
F4D4: LDA M2,X
F4D5: 90 02
F4D6: BCC DIV4
F4D7: IF M2<E THEN DON'T RESTORE M2.
F4D8: A2 FD
F4D9: LDX
F4DA: #$FD
F4DB: INDEX FOR 3-BYTE CONDITIONAL MOVE
F4DC: 68
F4DD: MD3
F4DE: PLA
F4DF: POP ONE RETURN LEVEL.
F4E0: 85 F9
F4E1: FIXRTS
F4E2: RTS
F4E3: CLEAR X1 AND RETURN.
F4E4: 49 80
F4E5: MD3
F4E6: EOR #$80
F4E7: COMPLEMENT SIGN BIT OF EXPONENT.
F4E8: 85 F8
F4E9: STA X1
F4EA: STORE IT.
F4EB: 20 7D F4
F4EC: FIX1
F4ED: JSR
F4EE: RTAR
F4EF: LDA X1
F4F0: 10 13
F4F1: BPL UNDPL
F4F2: 86 PF
F4F3: MD2
F4F4: STX M1+2
F4F5: CLEAR MANT1 (3 BYTES) FOR MUL/DIV.
F4F6: 86 PF
F4F7: STX M1+1
F4F8: CLEAR MANT1 (3 BYTES) FOR MUL/DIV.
F4F9: 86 F9
F4FA: STX M1
F4FB: 80 OD
F4FC: BS
F4FD: CSVCHK
F4FE: IF CALC. SET CARRY,CHECK FOR OVFL
F4FF: 30 04
F4G0: BMI MD3
F4G1: IF NEG THEN NO UNDERFLOW.
F4G2: 68
F4G3: PLA
F4G4: POP ONE RETURN LEVEL.
F4G5: 68
F4G6: PLA
F4G7: 90 B2
F4G8: BCC NORMX
F4G9: CLEAR X1 AND RETURN.
F4GA: 49 80
F4GB: MD3
F4GC: EOR #$80
F4GD: COMPLEMENT SIGN BIT OF EXPONENT.
F4GE: 85 F8
F4GF: STA X1
F4G0: STORE IT.
F4G1: 20 17
F4G2: LDY
F4G3: #$17
F4G4: COUNT 24 MUL/23 DIV ITERATIONS.
F4G5: 60
F4G6: RTS
F4G7: RETURN.
F4G8: 10 F7
F4G9: OVCAN
F4GA: BPL MD3
F4GB: IF POSITIVE EXP THEN NO OVFL.
F4GC: 4C F5
F4GD: 03 OVCAN
F4GE: JMP OVCAN
F4GF: ORG
F4G0: #$F63D
F4G1: P63D
F4G2: 20 7D F4
F4G3: FIX1
F4G4: JSR
F4G5: RTAR
F4G6: A5 F8
F4G7: FIX
F4G8: LDA X1
F4G9: F642: 10 13
F4GA: BPL UNDPL
F4GB: 86 PF
F4GC: MD2
F4GD: STX M1+2
F4GE: 86 PF
F4GF: STX M1+1
F4G0: CLEAR MANT1 (3 BYTES) FOR MUL/DIV.
F4G1: 86 F9
F4G2: STX M1
F4G3: 80 OD
F4G4: BS
F4G5: CSVCHK
F4G6: IF CALC. SET CARRY,CHECK FOR OVFL
F4G7: 30 04
F4G8: BMI MD3
F4G9: IF NEG THEN NO UNDERFLOW.
F4GA: 68
F4GB: PLA
F4GC: POP ONE RETURN LEVEL.
F4GD: 68
F4GE: PLA
F4GF: 90 B2
F4G0: BCC NORMX
F4G1: CLEAR X1 AND RETURN.
F4G2: 49 80
F4G3: MD3
F4G4: EOR #$80
F4G5: COMPLEMENT SIGN BIT OF EXPONENT.
F4G6: 85 F8
F4G7: STA X1
F4G8: STORE IT.
***********************

APPLE-II PSEUDO MACHINE INTERPRETER

COPYRIGHT 1977 APPLE COMPUTER INC

ALL RIGHTS RESERVED

S. WOZNIAK

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

TITLE "SWEET16 INTERPRETER"

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

F689: 20 4A FF  SW16     JSR   SAVE       PRESERVE 6502 REG CONTENTS
F68C: 68     PLA
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
F695: 4C 92 F6           JMP   SW16B      ONE SWEET16 INSTR.
F698: E6 1E     INC   R15L
F69A: D0 02     BNE   SW16D      INCR SWEET16 PC FOR FETCH
F69C: E6 1F     INC   R15H
F69E: A9 F7  SW16D    LDA   #SW16PAG
F6A0: 48     PHA              PUSH ON STACK FOR RTS
F6A3: A0 00     LOY   #$0
F6A5: 29 0F     AND   #$F        MASK REG SPECIFICATION
F6A7: 0A     ASL   A          DOUBLE FOR TWO BYTE REGISTERS
F6A8: AA     TAX              TO X REG FOR INDEXING
F6AA: 4A     LSR   A
F6AB: 51 1E     EOR   (R15L),Y   NOW HAVE OPCODE
F6AC: F0 0B     BEQ   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 F1 P6    LDA   OPTBL-2,Y   LOW ORDER ADR BYTE
F6B7: 48     PHA   ONTO STACK
F6BB: 60     RTS              GOTO REG-OP ROUTINE
F6BF: 6E 1E     TOBR     INC   R15L
F6C2: 4A     LSR   A          PREPARE CARRY FOR BC, BNC.
F6C3: A5 1D     INC   R14H       'PRIOR RESULT REG' INDEX
F6C5: 4A     LSR   A
F6C6: 60     RTS              GOTO NON-REG OP ROUTINE
F6C7: 68     RTNZ  PLA              POP RETURN ADDRESS
F6C8: 68     PLA
F6CD: 20 3F PP     JSR   RESTORE RESTORE 6502 REG CONTENTS
F6CC: 6C 1E 00     JMP   (R15L)   RETURN TO 6502 CODE VIA PC
F6CF: B1 1K     SETZ   LDA   (R15L),Y   HIGH-ORDER BYTE OF CONSTANT

96
F755:  A5  01            LDA  R0H          BYTE AND INCR RX. THEN
F757:  81  00            STA  (R0L,X)     STORE R0 LOW BYTE $E0
F759:  4C  1F  F7        JMP  INR          INCR RX AND RETURN
F75C:  20  66  F7        STPAT  DCR2       DCR RX
F75F:  A5  00            LDA  R0L          STORE HIGH-ORDER BYTE.
F761:  81  00            STA  (R0L,X)     STORE RX
F763:  4C  43  F7        JMP  INR          INCR RX AND RETURN
F766:  B5  00            DCR  R0L,X
F768:  D0  02            BNE  DCR2       DCR RX
F76A:  D6  01            DEC  R0H,X
F76C:  D6  00  DCR2       DEC  R0L,X
F76E:  60               RTS
F76F:  A0  00  SUB        LDY  #$0          RESULT TO R0
F771:  38  CPR           SBC              NOTE Y-REG = 13*2 FOR CPR
F772:  A5  00            LDA  R0L          LDA R0L AND INC RX
F774:  F5  00            SBC  R0L,X        STORE LDA AS R0H
F776:  99  00  00        STA  R0L,Y      R0-RX TO RY
F778:  A5  01            LDA  R0H          LDA R0H
F77A:  D0  02            BNE  DCR2       DCR RX
F77C:  20  66  F7        STPAT  DCR2       DCR RX
F77E:  A0  00  SUB        LDY  #$0          RESULT TO R0
F781:  69  00  ADD        LDA  R0L,X      ADD RX TO R0L
F783:  85  1D            STA  R14H        LAST RESULT REG*2
F785:  60               RTS
F786:  A5  00  ADD        LDA  R0L          ADD RX TO R0L
F788:  75  00  ADD        LDA  R0L,X       ADD RX TO R0L WITH INDEX
F78A:  85  00            STA  R0L          ADD TO R0L
F78C:  A5  01            LDA  R0H          ADD TO R0H
F78E:  75  01  ADD        LDA  R0H,X       ADD TO R0H,X
F790:  A0  00            LDX  #$20        R0 FOR RESULT
F792:  F0  E9            BEQ  SUB2       FINISH ADD
F794:  A5  1B  BS        LDA  R15L,Y     NOTE X-REG IS 12*2!
F796:  20  19  F7        JSR  STAT2      PUSH LOW ORDER PC VIA R12
F798:  A5  1F            LDA  R15H
F79A:  20  19  F7        JSR  STAT2      PUSH HIGH-ORDER PC BYTE
F79E:  18  CLC           BR
F79F:  A0  00  BNC        BCS  BNC2      NO CARRY TEST
F7A1:  A1  1B  BR1       LDA  (R15L),Y  DISPLACEMENT BYTE
F7A3:  10  01  BPL        ADD TO PC
F7A5:  88               DEY
F7A6:  65  1E  BR2       ADC  R15L      ADD TO PC
F7A8:  85  1E            STA  R15L
F7AA:  98               TYA
F7AB:  65  1F  ADC       R15H
F7AD:  85  1F            STA  R15H
F7AF:  60               BNC2          RTS
F7B0:  8C  BC  BCS       BR
F7B2:  60               RTS
F7B3:  0A  BP          ASL  A          DOUBLE RESULT-REG INDEX
F7B4:  AA              TAX              TO X REG FOR INDEXING
F7B5:  B5  01           LDA  R0H,X      TEST FOR PLUS
F7B7:  10  B8           BPL  BR1       BRANCH IF SO
F7B9:  60               RTS
F7BA:  0A  BM          ASL  A          DOUBLE RESULT-REG INDEX
F7BB:  AA              TAX
F7BC:  B5  01           LDA  R0H,X      TEST FOR MINUS
F7BE:  30  EL           BMI  BR1       BRANCH IF SO
F7C0:  60               RTS
F7C1:  0A  BZ          ASL  A          DOUBLE RESULT-REG INDEX
F7C2:  AA              TAX
F7C3:  B5  00           LDA  R0L,X      TEST FOR ZERO
F7C5:  15  01           ORA  R0H,X      (BOTH BYTES)
F7C7:  F0  D8           BEQ  BR1       BRANCH IF SO
F7C9:  60               RTS
F7CA:  0A  BNZ         ASL  A          DOUBLE RESULT-REG INDEX
F7CB:  AA              TAX
F7CC:  B5  00           LDA  R0L,X      TEST FOR NON-ZERO
F7CE:  15  01           ORA  R0H,X      (BOTH BYTES)
F7D0:  D0  CF           BNE  BR1       BRANCH IF SO
F7D2:  60               RTS
F7D3:  0A  BM1         ASL  A          DOUBLE RESULT-REG INDEX
F7D4:  AA              TAX
F7D5:  B5  00           LDA  R0L,X      CHECK BOTH BYTES
F7D7:  35  01           AND  R0H,X      FOR $FF (MINUS 1)
F7D9:  49  FF           EOR  #$FF
F7DB:  F0  C4           BEQ  BR1       BRANCH IF SO
F7DD:  60               RTS
F7DE:  0A  BNM1        ASL  A          DOUBLE RESULT-REG INDEX
F7DF:  AA              TAX
F7E0:  B5  00           LDA  R0L,X      CHECK BOTH BYTES FOR NO $FF
F7E2:  35  01           AND  R0H,X      CHECK BOTH BYTES FOR NO $FF
F7E4:  49  FF           EOR  #$FF
F7E6:  D0  B9           BNE  BR1       BRANCH IF NOT MINUS 1
F7E8:  60               RTS
F7E9:  A2  18  R8        LDX  #$18  12*2 FOR R12 AS STACK POINTER

98
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

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

A
X,Y
M
C
P
S
✓
−
+
∧
安东  
Logical AND  
Subtract  
Logical Exclusive OR  
Transfer From Stack  
Transfer To Stack  
Transfer To  
Transfer To  
V
Logic OR  
PC
PCH
PCL
OPER
#  
Immediate Addressing Mode

ACCUMULATOR
INDEX REGISTER Y
INDEX REGISTER X
PROGRAM COUNTER
STACK POINTER
PROCESSOR STATUS REGISTER, "P"

NOTE 1: BIT — TEST BITS
Bit 6 and 7 are transferred to the status register. If the result of A ∧ M is zero than Z=1, otherwise Z=0.
### 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>ADC</td>
<td>A • M • C — A C</td>
<td>Immediate</td>
<td>ADC #0per</td>
<td>69 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>ADC Oper.X</td>
<td>62 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>ADC Oper.X</td>
<td>63 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>ADC Oper.X</td>
<td>64 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect,X)</td>
<td>ADC Oper.X</td>
<td>65 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect,Y)</td>
<td>ADC Oper.X</td>
<td>71 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add memory to accumulator with carry</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>A • M — A</td>
<td>Immediate</td>
<td>AND #0per</td>
<td>26 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>AND Oper.X</td>
<td>25 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>AND Oper.X</td>
<td>24 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>AND Oper.Y</td>
<td>23 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect,X)</td>
<td>AND (Oper,X)</td>
<td>21 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect,Y)</td>
<td>AND (Oper,Y)</td>
<td>31 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AND memory with accumulator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASL</td>
<td>(See Figure 1)</td>
<td>Accumulator</td>
<td>ASL A</td>
<td>0A 1</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>ASL Oper.X</td>
<td>1E 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>ASL Oper.X</td>
<td>16 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>ASL Oper.Y</td>
<td>15 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect,X)</td>
<td>ASL (Oper,X)</td>
<td>14 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect,Y)</td>
<td>ASL (Oper,Y)</td>
<td>31 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shit left one bit (Memory or Accumulator)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCC</td>
<td>Branch on carry clear</td>
<td>Branch on C=0 Relative</td>
<td>BCC Oper</td>
<td>90 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>BCS</td>
<td>Branch on carry set</td>
<td>Branch on C=1 Relative</td>
<td>BCS Oper</td>
<td>81 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>BEQ</td>
<td>Branch on result zero</td>
<td>Branch on Z=1 Relative</td>
<td>BEQ Oper</td>
<td>81 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td>Test bits in memory with accumulator</td>
<td>A • M, M_{Z} — M, Zero Page X, Absolute</td>
<td>BIT Oper</td>
<td>2A 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>BIT Oper</td>
<td>29 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect,X)</td>
<td>BIT (Oper,X)</td>
<td>27 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect,Y)</td>
<td>BIT (Oper,Y)</td>
<td>31 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Branch on result minus</td>
<td>Branch on N=1 Relative</td>
<td>BMI Oper</td>
<td>30 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>BNE</td>
<td>Branch on result not zero</td>
<td>Branch on Z=0 Relative</td>
<td>BNE Oper</td>
<td>80 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>BPL</td>
<td>Branch on result plus</td>
<td>Branch on N=0 Relative</td>
<td>BPL Oper</td>
<td>10 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>BRK</td>
<td>Branch on overflow clear</td>
<td>Branch on V=0 Relative</td>
<td>BRK* Oper</td>
<td>00 1</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>BVC</td>
<td>Branch on overflow clear</td>
<td>Branch on V=0 Relative</td>
<td>BVC Oper</td>
<td>50 2</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
</tbody>
</table>
## INSTRUCTION CODES

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX OP Code</th>
<th>No. Bytes</th>
<th>'P' Status Reg MZCIBY</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOR (Exclusive-Or memory with accumulator)</td>
<td>A VM → A</td>
<td>Immediate</td>
<td>EOR $0</td>
<td>0E</td>
<td>2</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>INC Increment memory by one</td>
<td>M + 1 → M</td>
<td>Zero Page, Zero Page, Absolute, Absolute</td>
<td>INC Oper</td>
<td>46</td>
<td>2</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>INX Increment index X by one</td>
<td>X + 1 → X</td>
<td>Indirect</td>
<td>INX</td>
<td>66</td>
<td>1</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>INY Increment index Y by one</td>
<td>Y + 1 → Y</td>
<td>Indirect</td>
<td>INY</td>
<td>68</td>
<td>1</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>JMP Jump to new location</td>
<td>(PC-1) → PCL, (PC) → PCH</td>
<td>Absolute, Indirect</td>
<td>JMP Oper, JMP (Oper)</td>
<td>4C</td>
<td>3</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>JSR Jump to new location saving return address</td>
<td>PC-2 + (PC-1) → PCL, (PC-2) → PCH</td>
<td>Absolute</td>
<td>JSR Oper</td>
<td>20</td>
<td>3</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>LDA Load accumulator with memory</td>
<td>M → A</td>
<td>Immediate</td>
<td>LDA $0</td>
<td>0A</td>
<td>2</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>LDX Load index X with memory</td>
<td>M → X</td>
<td>Immediate</td>
<td>LDX $0</td>
<td>0A</td>
<td>2</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>LDY Load index Y with memory</td>
<td>M → Y</td>
<td>Immediate</td>
<td>LDY $0</td>
<td>0A</td>
<td>2</td>
<td>&quot;---&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX OP Code</th>
<th>No. Bytes</th>
<th>'P' Status Reg MZCIBY</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP No operation</td>
<td>No Operation</td>
<td>Implied</td>
<td>NOP</td>
<td>EA</td>
<td>1</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>LSR Shift right one bit (memory or accumulator)</td>
<td>A VM → A</td>
<td>Immediate</td>
<td>LSR $0</td>
<td>4A</td>
<td>1</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>LSR Shift right one bit (memory or accumulator)</td>
<td>A VM → A</td>
<td>Immediate</td>
<td>LSR $0</td>
<td>4A</td>
<td>1</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>LSR Shift right one bit (memory or accumulator)</td>
<td>A VM → A</td>
<td>Immediate</td>
<td>LSR $0</td>
<td>4A</td>
<td>1</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>LSR Shift right one bit (memory or accumulator)</td>
<td>A VM → A</td>
<td>Immediate</td>
<td>LSR $0</td>
<td>4A</td>
<td>1</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>LSR Shift right one bit (memory or accumulator)</td>
<td>A VM → A</td>
<td>Immediate</td>
<td>LSR $0</td>
<td>4A</td>
<td>1</td>
<td>&quot;---&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX OP Code</th>
<th>No. Bytes</th>
<th>'P' Status Reg MZCIBY</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHA Push accumulator on stack</td>
<td>A</td>
<td>Implied</td>
<td>PHA</td>
<td>48</td>
<td>1</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>PHP Push processor status on stack</td>
<td>P</td>
<td>Implied</td>
<td>PHP</td>
<td>08</td>
<td>1</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>PLA Pull accumulator from stack</td>
<td>A</td>
<td>Implied</td>
<td>PLA</td>
<td>68</td>
<td>1</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>PLP Pull processor status from stack</td>
<td>P</td>
<td>Implied</td>
<td>PLP</td>
<td>28</td>
<td>1</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>ROL Rotate one bit left (memory or accumulator)</td>
<td>(See Figure 2)</td>
<td>Accumulator</td>
<td>ROL $0</td>
<td>2A</td>
<td>1</td>
<td>&quot;---&quot;</td>
</tr>
<tr>
<td>ROR Rotate one bit right (memory or accumulator)</td>
<td>(See Figure 3)</td>
<td>Accumulator</td>
<td>ROR $0</td>
<td>2A</td>
<td>1</td>
<td>&quot;---&quot;</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 Code</th>
<th>No. Bytes</th>
<th>P Status Reg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTI</td>
<td>Return from interrupt</td>
<td>F1, PC1</td>
<td>Implied</td>
<td>RTI</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>RTS</td>
<td>Return from subroutine</td>
<td>PC1, PC1→PC</td>
<td>Implied</td>
<td>RTS</td>
<td>1</td>
<td>-----</td>
</tr>
<tr>
<td>SBC</td>
<td>Subtract memory from accumulator with borrow</td>
<td>A·M·C→A</td>
<td>Immediate</td>
<td>SBC #0</td>
<td>E0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>SBC Oper</td>
<td>E5</td>
<td>2</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>SBC Oper</td>
<td>F6</td>
<td>2</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute.X</td>
<td>SBC Oper</td>
<td>F9</td>
<td>3</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute.Y</td>
<td>SBC Oper</td>
<td>E1</td>
<td>2</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect)X</td>
<td>SBC (Oper,x)</td>
<td>F1</td>
<td>2</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect)Y</td>
<td>SBC (Oper,y)</td>
<td>F1</td>
<td>2</td>
<td>------</td>
</tr>
<tr>
<td>SEC</td>
<td>Set carry flag</td>
<td>1→C</td>
<td>Implied</td>
<td>SEC</td>
<td>38</td>
<td>1</td>
</tr>
<tr>
<td>SED</td>
<td>Set decimal mode</td>
<td>1→D</td>
<td>Implied</td>
<td>SED</td>
<td>F6</td>
<td>1</td>
</tr>
<tr>
<td>SEI</td>
<td>Set interrupt disable status</td>
<td>1→I</td>
<td>Implied</td>
<td>SEI</td>
<td>78</td>
<td>1</td>
</tr>
<tr>
<td>STA</td>
<td>Store accumulator in memory</td>
<td>A→M</td>
<td>Zero Page</td>
<td>STA Oper</td>
<td>86</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page.X</td>
<td>STA Oper</td>
<td>96</td>
<td>2</td>
<td>-----</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>STA Oper</td>
<td>96</td>
<td>3</td>
<td>-----</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute.X</td>
<td>STA Oper</td>
<td>96</td>
<td>3</td>
<td>-----</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute.Y</td>
<td>STA Oper</td>
<td>96</td>
<td>3</td>
<td>-----</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect)X</td>
<td>STA Oper,x</td>
<td>96</td>
<td>3</td>
<td>-----</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect)Y</td>
<td>STA Oper,y</td>
<td>96</td>
<td>3</td>
<td>-----</td>
</tr>
<tr>
<td>STX</td>
<td>Store index X in memory</td>
<td>X→M</td>
<td>Zero Page</td>
<td>STX Oper</td>
<td>86</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page.Y</td>
<td>STX Oper</td>
<td>96</td>
<td>2</td>
<td>-----</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>STX Oper</td>
<td>86</td>
<td>3</td>
<td>-----</td>
</tr>
<tr>
<td>STY</td>
<td>Store index Y in memory</td>
<td>Y→M</td>
<td>Zero Page</td>
<td>STY Oper</td>
<td>86</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page.X</td>
<td>STY Oper</td>
<td>94</td>
<td>2</td>
<td>-----</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>STY Oper</td>
<td>94</td>
<td>3</td>
<td>-----</td>
</tr>
<tr>
<td>TAX</td>
<td>Transfer accumulator to index X</td>
<td>A→X</td>
<td>Implied</td>
<td>TAX</td>
<td>AA</td>
<td>1</td>
</tr>
<tr>
<td>TAY</td>
<td>Transfer accumulator to index Y</td>
<td>A→Y</td>
<td>Implied</td>
<td>TAY</td>
<td>AB</td>
<td>1</td>
</tr>
<tr>
<td>TSX</td>
<td>Transfer stack pointer to index X</td>
<td>S→X</td>
<td>Implied</td>
<td>TSX</td>
<td>8A</td>
<td>1</td>
</tr>
</tbody>
</table>

### TXA
- Transfer index X to accumulator
  - Operation: X→A
  - Addressing Mode: Implied
  - Assembly Language Form: TXA
  - HEX Code: 8A
  - No. Bytes: 1
  - P Status Reg.: ------

### TXS
- Transfer index X to stack pointer
  - Operation: X→S
  - Addressing Mode: Implied
  - Assembly Language Form: TXS
  - HEX Code: 9A
  - No. Bytes: 1
  - P Status Reg.: ------

### TYA
- Transfer index Y to accumulator
  - Operation: Y→A
  - Addressing Mode: Implied
  - Assembly Language Form: TYA
  - HEX Code: 98
  - No. Bytes: 1
  - P Status Reg.: ------
HEX OPERATION CODES

00 — BRK
01 — ORA — (Indirect, XI)
02 — NOP
03 — NOR
04 — NOP
05 — ORA — Zero Page
06 — ASL — Zero Page
07 — NOP
08 — PHP
09 — ORA — Immediate
0A — ASL — Accumulator
0B — NOP
0C — NOP
0D — ORA — Absolute
0E — ASL — Absolute
0F — NOP
10 — BPL
11 — ORA — (Indirect), Y
12 — NOP
13 — NOP
14 — NOR
15 — ORA — Zero Page, X
16 — ASL — Zero Page, X
17 — NOR
18 — CLC
19 — ORA — Absolute, Y
1A — NOR
1B — NOP
1C — NOR
1D — ORA — Absolute, X
1E — ASL — Absolute, X
1F — NOP
20 — JSR
21 — AND — (Indirect, X)
22 — NOR
23 — NOP
24 — BIT — Zero Page
25 — AND — Zero Page
26 — ROL — Zero Page
27 — NOP
28 — PLP
29 — AND — Immediate
2A — ROL — Accumulator
2B — NOP
2C — BIT — Absolute
2D — AND — Absolute
2E — ROL — Absolute
2F — NOP
30 — BPI
31 — AND — (Indirect), V
32 — NOP
33 — NOP
34 — NOP
35 — AND — Zero Page, X
36 — ROL — Zero Page, X
37 — NOP
38 — CLC
39 — ORA — Absolute, Y
3A — NOP
3B — NOP
3C — NOP
3D — AND — Absolute, X
3E — ROL — Absolute, X
3F — NOP
40 — JSR
41 — AND — Indirect, X
42 — NOP
43 — NOP
44 — NOP
45 — ROR — Zero Page
46 — ROR — Zero Page
47 — NOP
48 — PHA
49 — AND — Immediate
4A — ROR — Accumulator
4B — NOP
4C — NOP
4D — ROR — Absolute
4E — ROR — Absolute
4F — NOP
50 — BM!
51 — AND — (Indirect), V
52 — NOP
53 — NOP
54 — NOP
55 — AND — Zero Page, X
56 — ROR — Zero Page, X
57 — NOP
58 — CLI
59 — FOR — Absolute, Y
60 — RTS
61 — ADC — Indirect, X
62 — ADC — Indirect, X
63 — NOP
64 — ADC — Zero Page
65 — ADC — Zero Page
66 — ROR — Zero Page
67 — NOP
68 — PLA
69 — ADC — Immediate
6A — ROR — Accumulator
6B — NOP
6C — JMP — Absolute
6D — ADC — Absolute
6E — ROR — Absolute
6F — NOP
70 — BCC
71 — STA — (Indirect), Y
72 — NOP
73 — STY — Zero Page
74 — STA — Zero Page
75 — STX — Zero Page
76 — NOP
77 — DEY
78 — TXA
79 — STY — Absolute
7A — TXS
7B — MOP
7C — NOP
7D — STA — Absolute
7E — 80H — Absolute, X
7F — NOP
80 — STY — Zero Page
81 — STA — (Indirect, Xi
82 — NOP
83 — NOP
84 — STY — Zero Page
85 — STA — Zero Page
86 — STX — Zero Page
87 — NOP
88 — DEY
89 — NOP
8A — TXA
8B — NOP
8C — STY — Absolute
8D — STA — Absolute
8E — STX — Absolute
8F — NOP
90 — BCC
91 — STA — Zero Page
92 — NOP
93 — NOR
94 — STA — Absolute, Y
95 — STA — Zero Page
96 — STX — Zero Page, Y
97 — NOP
98 — TAY
99 — STA — Absolute, Y
9A — NOP
9B — CLV
9C — NOP
9D — STA — Absolute
9E — NOP
9F — NOP
A0 — LDY — Immediate
A1 — LDA — (Indirect, X)
A2 — LDA — Immediate
A3 — NOP
A4 — LDY — Immediate
A5 — LDA — Absolute
A6 — LDX — Absolute
A7 — NOP
A8 — TAY
A9 — LDA — Immediate
AA — TAX
AB — NOP
AC — LDA — Absolute
AD — LDA — Absolute
AE — LDX — Absolute
AF — NOP
B0 — BCS
B1 — CMP — (Indirect), X
B2 — CMP — (Indirect, X)
B3 — CMP — Absolute
B4 — CMP — Absolute
B5 — CMP — Absolute
B6 — CMP — Absolute
B7 — NOP
B8 — CLV
B9 — STA — Absolute
BA — STY — Absolute
BB — NOP
BC — LDY — Absolute, X
BD — LDA — Absolute, X
BE — LOX — Absolute, Y
BF — NOP
C0 — CPY — Immediate
C1 — CMP — Immediate
C2 — CMP — Immediate
C3 — NOP
C4 — CPY — Zero Page
C5 — CMP — Zero Page
C6 — DEC — Zero Page
C7 — NOP
C8 — INY
C9 — CMP — Immediate
CA — DEX
CB — MOP
CC — CPY — Absolute
CD — CMP — Absolute
CE — DEC — Absolute
CF — NOP
D0 — BNE
D1 — CMP — (Indirect), Y
D2 — NOP
D3 — NOP
D4 — NOP
D5 — CPY — Zero Page
D6 — DEC — Zero Page
D7 — NOP
D8 — NOP
D9 — CMP — Absolute, Y
DA — NOP
DB — MOP
DC — CMP — Absolute
DD — CMP — Absolute
DE — DEC — Absolute
DF — NOP
E0 — CPX — Immediate
E1 — SBC — (Indirect, X)
E2 — NOP
E3 — NOP
E4 — CPX — Zero Page
E5 — SBC — (Indirect, X)
E6 — INC — Zero Page
E7 — NOP
E8 — NOP
E9 — SBC — Immediate
EA — NOP
EB — NOP
EC — CPX — Absolute
ED — SBC — Absolute
EE — INC — Absolute
EF — NOP
F0 — BM
F1 — SBC — (Indirect, Y)
F2 — NOP
F3 — NOP
F4 — NOP
F5 — SBC — Zero Page, X
F6 — INC — Zero Page, X
F7 — NOP
F8 — SED
F9 — SBC — Absolute, Y
FA — NOP
FD — NOP
FE — INC — Absolute, X
FF — NOP

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 WmA.

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

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

k. Cable for the following:

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

Optionally you may desire:

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

m. Case to hold all the above

Final Assembly Steps

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

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

3. Plug in speaker cable.

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

5. Connect video cable.

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

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

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

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

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

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

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

RUNNING BASIC

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The power supply is allowed to run 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.
Accessories are available to aid the user in connecting the Apple II system to a home color TV with a minimum of trouble. These units are called "RF Modulators" and they generate a radio frequency signal corresponding to the carrier of one or two of the lower VHF television bands: 61.25 MHz (channel 3) or 67.25 MHz (channel 4). This RF signal is then modulated with the composite video signal generated by the Apple II.

Users report success with the following RF modulators:

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

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

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

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

Most of the above are available through local computer stores.

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

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

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

Some cautions are in order. The Apple II, by virtue of its color graphics capability, operates the TV set in a linear mode rather than the 100% contrast mode satisfactory for displaying text. For this reason, radio frequency interference (RFI) generated by a computer (or peripherals) will beat with the
carrier of the RF modulator to produce faint spurious background patterns (called "worms"). This RFI "trash" must be of quite a low level if worms are to be prevented. In fact, these spurious beats must be 40 to 50 db below the signal level to reduce worms to an acceptable level. When it is remembered that only 2 to 6 mV (across 300Ω, is presented to the VHF input of the TV set, then stray RFI getting into the TV must be less than 500 μV to obtain a clean picture. Therefore we recommend that a good, co-ax cable be used to carry the signal from any modulator to the TV set, such as RG/59u (with copper shield), Belden #8241 or an equivalent miniature type such as Belden #8218. We also recommend that the RF modulator been 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 co-ax 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.
A SIMPLE SERIAL OUTPUT

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 la).
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 $370 ($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 $3700 activates the printer and hitting the "RESET" key exits the program.

Saving the Machine Language Program

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

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

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

3. Type in "370.03E9W 370.03E9W".

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

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

The Program

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

2. Turn the teletype (printer on)

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

4. Type in RUN and hit the "RETURN" key. The text in line 15 should be printed on the teletype and control is returned to the keyboard and Video monitor
Line 1Ø activates the teletype machine routine and all "PRINT" statements following it will be printed to the teletype until a PR#Ø statement is encountered. Then the text in line 15 will appear on the teletype's output. Line 2Ø deactivates the printer and the program ends on line 25.

Conclusion

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

(A)  (B)

FIGURE 2  ASR-33

FIGURE 2  RS-232
**TELETYPING DRIVER ROUTINES**

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

1 TITLE TELETYPING DRIVER ROUTINES'
2 ********************************************
3 *                                              *
4 *      TTYDRIVER:   *                        *
5 *      TELETYPING OUTPUT   *                 *
6 *      ROUTINE FOR 72   *                    *
7 *      COLUMN PRINT WITH  *                 *
8 *      BASIC LIST       *                    *
9 *                                              *
10 *      COPYRIGHT 1977 BY:   *                *
11 *      APPLE COMPUTER INC.    *            *
12 *      11/18/77    *                      *
13 *                                              *
14 *      R. WIGGINSTON   *                    *
15 *      S. WOZNIAK  *                       *
16 *                                              *
17 ********************************************
18 WNDWDTH EQU $21  ;FOR APPLE-II
19 CH EQU $24   ;CURSOR HORIZ.
20 CSWL EQU $36  ;CHAR. OUT SWITCH
21 YSAVE EQU $778
22 COLCNT EQU $7F8 ;COLUMN COUNT LOC.
23 MARK EQU $CO58
24 SPACE EQU $CO59
25 WAIT EQU $FCA8
26 ORG $370

***WARNING: OPERAND OVERFLOW IN LINE 27

0370: A9 82 27 TTINIT: LDA #TTOUT
0372: 85 36 28 STA CSWL  ;POINT TO TTY ROUTINES
0374: A9 03 29 LDA #TTOUT/256 ;HIGH BYTE
0376: 85 37 30 STA CSWL+1
0378: A9 48 31 LDA #72  ;SET WINDOW WIDTH
037A: 85 21 32 STA WNDWDTH ;TO NUMBER COLUMNS ONT
037C: A5 24 33 LDA CH
037E: 8D F8 34 STA COLCNT ;WHERE WE ARE NOW.
0381: 60 35 RTS
0382: 48 36 TTOUT: PHA  ;SAVE TWICE
0383: 48 37 PHA  ;ON STACK.
0384: AD F8 38 TTOUT2: LDA COLCNT ;CHECK FOR A TAB.
0387: C5 24 39 CMP CH
0389: 68 40 PLA  ;RESTORE OUTPUT CHAR.
038A: BO 03 41 BCS TESTCTRL  ;IF C SET, No TAB
038C: 48 42 PHA
038D: A9 AO 43 LDA #$A0  ;PRINT A SPACE.
038F: 2C CO 44 TESTCTRL:BIT RTS1  ;TRICK TO DETERMINE
0392: FO 03 45 BEQ PRNTIT  ;IF CONTROL CHAR.
0394: EE F8 46 INC COLCNT  ;IF NOT, ADD ONE TO CM
0397: 20 C1 47 PRNTIT: JSR DOCHAR  ;PRINT THE CHAR ON TTY
0399: 68 48 PLA  ;RESTORE CHAR
039A: 48 49 PHA TTOUT2  ;AND PUT BACK ON STAC
039C: 90 E6 50 BCC #$OD  ;DO MORE SPACES FOR TA
039E: 49 OD 51 FOR A  ;CHECK FOR CAR RET.
03A0: OA 52 ASL FINISH  ;ELIM PARITY
03A1: DO OD 53 BNE  ;IF NOT CR, DONE.

**FIGURE 3a**
**TELETYPE DRIVER ROUTINES**

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

STA COLCNT
LDA #38A
JSR DOCHAR
LDA #153
JSR 7AIT
LDA COLCNT
FINISH:
BCC RETURN
ADC #11F
PHA
LDA 3E0
S3C
SSC
BCC RETURN
ADC RETURN
PHA
BNE LDY
LDA 3CC
LDA 3E0
LSR
SBC
3NE
PLA
ROR
DEY
BNE
LDY
PLP
RTS

RETURN:
RTS
RTS1:
SETCH:
DOCHAR:
TTOUT3:
MARKOUT:
TTOUT4:
DLY1:
DLY2:

***SUCCESSFUL ASSEMBLY: NO ERRORS***

**FIGURE 3b**
CROSS-REFERENCE: TELETYPEx DRIVER ROUTINES

CH 0024 0033 0039 0065
COLCNT 0718 0034 0038 0046 0054 0059
0SYL 0036 0028 0030
DLY1 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
SEATCH 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
WNDWIDTH 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
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. \( V_{\text{IN}} = 1\text{Vpp (nominal), } Z_{\text{IN}} = 12\text{K Ohms.} \) 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. \( V_{\text{OUT}} = 25\text{ mV into 17 Ohms, } Z_{\text{OUT}} = 100\text{ Ohms.} \) 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 Ø 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 2Ø on all peripheral connectors will go low during Ø, of a read or write to any address C800-0FFF. This line will drive a total of 4 standard TTL loads.

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

NC: No connection.

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

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

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

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

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

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

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

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

+12V: Positive 12-Volt supply.

+5V: Positive 5-Volt supply

-5V: Negative 5-Volt supply.

-12V: Negative 12-Volt supply.

POWER CONNECTOR

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

PIN DESCRIPTION

GND: (2 pins) system circuit ground. 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_

**Top View**

_Toward Front Edge of PC Board_

**Pinout**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+12V</td>
</tr>
<tr>
<td>2</td>
<td>-12V</td>
</tr>
<tr>
<td>3</td>
<td>-5V</td>
</tr>
<tr>
<td>4</td>
<td>+5V</td>
</tr>
<tr>
<td>5-10</td>
<td>N.C.</td>
</tr>
<tr>
<td>11</td>
<td>USER 1</td>
</tr>
<tr>
<td>12-20</td>
<td>Device Select</td>
</tr>
<tr>
<td>21</td>
<td>INT IN</td>
</tr>
<tr>
<td>22</td>
<td>DMA IN</td>
</tr>
<tr>
<td>23</td>
<td>INH</td>
</tr>
<tr>
<td>24-25</td>
<td>DMA OUT</td>
</tr>
<tr>
<td>26</td>
<td>INT OUT</td>
</tr>
<tr>
<td>27</td>
<td>NMI</td>
</tr>
<tr>
<td>28</td>
<td>IRQ</td>
</tr>
<tr>
<td>29</td>
<td>RES</td>
</tr>
<tr>
<td>30</td>
<td>R/W</td>
</tr>
<tr>
<td>31</td>
<td>I/O STROBE</td>
</tr>
<tr>
<td>32</td>
<td>N.C.</td>
</tr>
<tr>
<td>33-36</td>
<td>-12V</td>
</tr>
<tr>
<td>37-39</td>
<td>-5V</td>
</tr>
<tr>
<td>40-49</td>
<td>N.C.</td>
</tr>
</tbody>
</table>

**Locations JS to J12**

### Figure 5

**Power Connector**

**Top View**

_Toward Right Side of PC Board_

**Pinout**

- **(Blue/White Wire) -12V**
- **(Orange Wire) +5V**
- **(Black Wire) GND**
- **-5V (Blue Wire)**
- **+12V (Orange/White Wire)**
- **GND (Black Wire)**

**Location K1**

130
SPEAKER CONNECTOR

This is a MOLEX KK 100 series connector with two .25" square pins on .10" centers. See location and pin out in Figures 1 and 6.

SIGNAL DESCRIPTION FOR SPEAKER

+5V: System +5 Volts

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

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. 0 Volt line from power supply.

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

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

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

Figure 7

AUXILIARY VIDEO OUTPUT CONNECTOR

PINOUT

+12V

-5V

VIDEO

GND

Back Edge of PC Board

Right Edge of PC Board

LOCATION J14B
INSTALLING YOUR OWN RAM

THE POSSIBILITIES

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

RESTRICTIONS

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

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

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

ASSIGNING RAM

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

Figure 8

MEMORY SELECT SOCKETS

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

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

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

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

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

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

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

HOW TO USE

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

Let us learn by example:

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>0000</th>
<th>text and color graphics display pages, 6502 stack, pointers, etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>0200</td>
<td>high res. graphics display primary page</td>
</tr>
<tr>
<td>0300</td>
<td></td>
</tr>
<tr>
<td>0400</td>
<td>high res. graphics display secondary page</td>
</tr>
<tr>
<td>0500</td>
<td></td>
</tr>
<tr>
<td>0600</td>
<td></td>
</tr>
<tr>
<td>0700</td>
<td></td>
</tr>
<tr>
<td>0800</td>
<td>addresses dedicated to hardware functions</td>
</tr>
<tr>
<td></td>
<td>ROM socket DO: spare</td>
</tr>
<tr>
<td></td>
<td>ROM socket DS: spare</td>
</tr>
<tr>
<td></td>
<td>ROM socket EO: BASIC</td>
</tr>
<tr>
<td></td>
<td>ROM socket ES: BASIC</td>
</tr>
<tr>
<td></td>
<td>ROM socket FO: BASIC</td>
</tr>
<tr>
<td></td>
<td>ROM socket FS: monitor</td>
</tr>
</tbody>
</table>

### Memory Map Pages 0 to BFF

<table>
<thead>
<tr>
<th>HEX ADDRESS(ES)</th>
<th>USED BY</th>
<th>USED FOR</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGE ZERO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000-001F</td>
<td>UTILITY</td>
<td></td>
<td>register area for &quot;sweet 16&quot; 16 bit firmware processor.</td>
</tr>
<tr>
<td>0020-004D</td>
<td>MONITOR</td>
<td></td>
<td>holds a 16 bit number that is randomized with each key entry.</td>
</tr>
<tr>
<td>004E-004F</td>
<td>MONITOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0050-0055</td>
<td>UTILITY</td>
<td></td>
<td>integer multiply and divide work space.</td>
</tr>
<tr>
<td>0055-00FF</td>
<td>BASIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00FF-000F</td>
<td>UTILITY</td>
<td></td>
<td>floating point work space.</td>
</tr>
<tr>
<td>PAGE ONE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100-01FF</td>
<td>6502</td>
<td>subroutine return stack.</td>
<td></td>
</tr>
<tr>
<td>PAGE TWO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0200-02FF</td>
<td></td>
<td>character input buffer.</td>
<td></td>
</tr>
<tr>
<td>PAGE THREE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03F8</td>
<td>MONITOR</td>
<td></td>
<td>Y (control Y) will cause a JSR to this location.</td>
</tr>
<tr>
<td>03FB</td>
<td></td>
<td></td>
<td>NMIs are vectored to this location.</td>
</tr>
<tr>
<td>03FE-03FF</td>
<td></td>
<td></td>
<td>IRQs are vectored to the address pointed to by these locations.</td>
</tr>
<tr>
<td>0400-07FF</td>
<td>DISPLAY</td>
<td></td>
<td>text or color graphics primary page.</td>
</tr>
<tr>
<td>0800-0BFF</td>
<td>DISPLAY</td>
<td></td>
<td>text or color graphics secondary page.</td>
</tr>
</tbody>
</table>

| BASIC initializes LOHEM to location 0800. |

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;CO40 STB&quot;</td>
<td>Output strobe to Game I/O connector.</td>
</tr>
<tr>
<td>C050</td>
<td>Set graphics mode</td>
<td></td>
</tr>
<tr>
<td>C051</td>
<td>&quot; text &quot;</td>
<td></td>
</tr>
<tr>
<td>C052</td>
<td>Set bottom 4 lines graphics</td>
<td></td>
</tr>
<tr>
<td>C053</td>
<td>&quot; &quot; &quot; &quot; text</td>
<td></td>
</tr>
<tr>
<td>C054</td>
<td>Display primary page</td>
<td></td>
</tr>
<tr>
<td>C055</td>
<td>&quot; secondary page</td>
<td></td>
</tr>
<tr>
<td>C056</td>
<td>Set high res. graphics</td>
<td></td>
</tr>
<tr>
<td>C057</td>
<td>&quot; color &quot;</td>
<td></td>
</tr>
<tr>
<td>C058</td>
<td>Clear &quot;ANO&quot;</td>
<td>Annunciator 0 output to Game I/O connector.</td>
</tr>
<tr>
<td>C059</td>
<td>Set &quot;</td>
<td></td>
</tr>
<tr>
<td>C05A</td>
<td>Clear &quot;AN1&quot;</td>
<td>Annunciator 1 output to Game I/O connector.</td>
</tr>
<tr>
<td>C05B</td>
<td>Set &quot;</td>
<td></td>
</tr>
<tr>
<td>C05C</td>
<td>Clear &quot;AN2&quot;</td>
<td>Annunciator 2 output to Game I/O connector.</td>
</tr>
<tr>
<td>C05D</td>
<td>Set &quot;</td>
<td></td>
</tr>
<tr>
<td>C05E</td>
<td>Clear &quot;AN3&quot;</td>
<td>Annunciator 3 output to Game I/O connector.</td>
</tr>
<tr>
<td>C05F</td>
<td>Set &quot;</td>
<td></td>
</tr>
<tr>
<td>HEX ADDRESS</td>
<td>ASSIGNED FUNCTION</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------</td>
<td>----------</td>
</tr>
<tr>
<td>C060/8</td>
<td>Cassette input</td>
<td>State of &quot;Cassette Data In&quot; appears in bit 7. Input on Game I/O connector appears in bit 7.</td>
</tr>
<tr>
<td>C061/9</td>
<td>&quot;SW1&quot;</td>
<td>State of Switch 1 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 2 input on Game I/O connector appears in bit 7.</td>
</tr>
<tr>
<td>C063/B</td>
<td>&quot;SW3&quot;</td>
<td>State of Switch 3 input on Game I/O connector appears in bit 7.</td>
</tr>
<tr>
<td>C064/C</td>
<td>Paddle 0 timer output</td>
<td>State of timer output for Paddle 0 appears in bit 7.</td>
</tr>
<tr>
<td>C065/D</td>
<td>&quot; 1 &quot; &quot; &quot;</td>
<td>State of timer output for Paddle 1 appears in bit 7.</td>
</tr>
<tr>
<td>C066/E</td>
<td>&quot; 2 &quot; &quot; &quot;</td>
<td>State of timer output for Paddle 2 appears in bit 7.</td>
</tr>
<tr>
<td>C067/F</td>
<td>&quot; 3 &quot; &quot; &quot;</td>
<td>State of timer output for Paddle 3 appears in bit 7.</td>
</tr>
<tr>
<td>C07X</td>
<td>&quot;PDL STB&quot;</td>
<td>Triggers paddle timers during $\phi_2$.</td>
</tr>
<tr>
<td>C08X</td>
<td>DEVICE SELECT 0</td>
<td>Pin 41 on the selected Peripheral Connector goes low during $\phi_2$.</td>
</tr>
<tr>
<td>C09X</td>
<td>&quot; 1 &quot;</td>
<td></td>
</tr>
<tr>
<td>C0A0</td>
<td>&quot; 2 &quot;</td>
<td></td>
</tr>
<tr>
<td>C0AX</td>
<td>&quot; 3 &quot;</td>
<td></td>
</tr>
<tr>
<td>C0BX</td>
<td>&quot; 4 &quot;</td>
<td></td>
</tr>
<tr>
<td>C0C0</td>
<td>&quot; 5 &quot;</td>
<td></td>
</tr>
<tr>
<td>C0DX</td>
<td>&quot; 6 &quot;</td>
<td></td>
</tr>
<tr>
<td>C0EX</td>
<td>&quot; 7 &quot;</td>
<td></td>
</tr>
<tr>
<td>C0FX</td>
<td>&quot; 8 &quot;</td>
<td></td>
</tr>
<tr>
<td>C10X</td>
<td>&quot; 9 &quot;</td>
<td>Expansion connectors.</td>
</tr>
<tr>
<td>C11X</td>
<td>&quot; A &quot;</td>
<td></td>
</tr>
<tr>
<td>C12X</td>
<td></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</td>
<td></td>
</tr>
<tr>
<td>C14X</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C15X</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C16X</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C17X</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C1XX</td>
<td>I/O SELECT 1</td>
<td>Pin 1 on the selected Peripheral Connector goes low during $\Phi_2$.</td>
</tr>
<tr>
<td>C2XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C3XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C4XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C5XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C6XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C7XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C8XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>C9XX</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>CA XX</td>
<td>A</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>CE XX</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.

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

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

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

Ø3: A general purpose timing signal which occurs at the same rate as the microprocessor clocks but is nonsymmetrical.

**MICROPROCESSOR OPERATIONS**

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

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

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

**SYSTEM TIMING DIAGRAM**

[System Timing Diagram Image]
FIGURE S-2  MPU AND SYSTEM BUS
FIGURE S-4  SYNC COUNTER
FIGURE S-5 ROM MEMORY
“D” SOURCES ARE FROM SYNC COUNT

FIG. S-4

SCREEN ADDRESS FROM SYNC COUNT

FIG. S-4

TO RAM ADDRESS LINES

FIG. S-8

*SEE FIG. S-6 FOR OTHER HALF OF C12

FIGURE S-7 RAM ADDRESS MUX
FIGURE S-8  4K TO 48K RAM MEMORY WITH DATA LATCH