### 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 .. 3
- 6. APPLE Integer BASIC ............ 3
- 7. Running Your First and Second Programs .... 3
- 8. Running 16K Startrek ......... 3
- 9. Loading a Program Tape ....... 4
- 10. Breakout and Color Demos Tapes .. 6
- 11. Breakout and Color Demos Program Listings .... 12
- 12. How to Play Startrek .......... 14
- 13. Loading HIRES Demo Tape .... 15

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

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

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

**13. Additional BASIC Program Examples**
- a. Rod's Color Pattern (4K) .... 55
- b. Pong (4K) ...................... 56
- c. Color Sketch (4K) ............ 57
- d. Mastermind (8K) .............. 59
- e. Biorhythm (4K) .............. 61
- f. Dragon Maze (4K) ............ 63
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 is in the "OFF" position. Connect the A.C. power cord to the Apple and to a 3 wire 120 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 anout 10 seconds when saving a program, so you normally have 6 1/2 seconds of leeway. If the playback volume is too high, you may pick up tape noise before getting to the set-up tone. Try a lower playback volume.

SYSTEM CHECKOUT

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

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

3. VIDEO DISPLAY INTERFACE -
   a) for a video monitor - check that a cable connects the monitor to the Apple's video output port.
   b) for a standard television - check that an adapter (RF modulator) is plugged into the Apple II (either in the video output (K 14) or the video auxillary socket (J148), and that a cable runs between the television and the Adapter's output socket.

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

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

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

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

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

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

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

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

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

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

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

10. Type in the word "RUN" and depress the "RETURN" key. The steps in loading a program have been completed and if everything has gone satisfactorily the program will be operating now.

LOADING PROBLEMS

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

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

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

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

Cassette Recorder Inadjustment

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

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

Often times new cassette recorders will not need this adjustment.

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

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

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

Dealing with the Loading Problem

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

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

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

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

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

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

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

UNSOLVED PROBLEMS

If you are having any unsolved loading problems, contact your nearest local dealer or Apple Computer Inc.
PROGRAM DESCRIPTION

Breakout is a color graphics game for the Apple II computer. The object of the game is to "knock-out" all 160 colored bricks from the playing field by hitting them with the bouncing ball. You direct the ball by hitting it with a paddle on the left side of the screen. You control the paddle with one of the Apple's Game Paddle controllers. But watch out: you can only miss the ball five times!

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

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

REQUIREMENTS

This program will fit into a 4K or greater system.

BASIC is the programming language used.

PLAYING BREAKOUT

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

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

COLOR DEMO TAPE

PROGRAM DESCRIPTION

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

REQUIREMENTS

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

PROGRAM LISTING

5 GOTO 15
10 G=POW (0)20)/6: IF G=0 THEN
0=0: IF G>34 THEN 0=34: COLOR=
0: VLIN 0,0;5 AT 0: COLOR=0;
IF P-> THEN 175: IF Q THEN
VLIN 0,0;1 AT 0:P=0: RETURN

15 DIM AX(15),BY(15);A=1;B=13;
C=9;D=6;E=15: TEXT = CALL -
936: YTAB 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/H?:Q:GRA CALL -
936: IF B(1,1)="H" THEN 40
: FOR I=0 TO 39: COLOR=I/2+
(132): VLIN I,29 AT I
30 NEXT I: POKES 34,28: PRINT
: PRINT = FOR I=0 TO 19:
VTAB 21+I MOD 2: TAB I+
141: PRINT 1; NEXT I: POKES
34,22: YTAB 24: PRINT = PRINT
"BACKGROUND";
35 GOSUB 95;POKES: PRINT "EVEN BRICK":
": GOSUB 95;POKES: PRINT "GOO GOO
K":GOSUB 95;POKES: PRINT "FROG":
": GOSUB 95;POKES: PRINT "BALL";
: GOSUB 95
40 POKES 34,20: COLOR=8: FOR I=0
TO 39: VLIN 0,39 AT I: NEXT
I: FOR I=0 TO 34 STEP 2: TAB
1+i1: PRINT I/2-9;: COLOR=8:
VLIN 0,39 AT 1: COLOR=0: FOR
J=1 MOD 4 TO 39 STEP 4
45 VLIN J,J+1 AT 1: NEXT J,1; TAB
5: PRINT "SCORE = 0": PRINT
: PRINT = POKE 34,21;S=0;F=
S=1:S=X=19;Y=19;L=6
50 COLOR=A: PLOT X,Y/3;X=3;C=
RND (128):V=-1;N=RND (5)-
L=1=L=1: IF L1 THEN 120: TAB
6: IF L>1 THEN PRINT L," BALLS L
LEFT"
55 IF L=1 THEN PRINT "LAST BALL,":
";A#:PRINT : FOR I=1 TO 100
: GOSUB 10: NEXT I=N+1;
60 J=V: IF J=0 AND J(120 THEN
65;W=W+4=W+8; FOR I=1 TO 6;X=
PEEK (-16336): NEXT 1
70 J=V+W: IF J>0 Then 100: GOSUB
170: COLOR=K=J/3: IF J>39
75: IF SCR(I,K)=A THEN
80: IF I THEN 100;W-125=V=(
N5)+1;w=(K-P)+2;N+1=
70 Z=PEEK (-16336)>PEEK (-16336)
)+PEEK (-16336)>PEEK (-16336)
)+(PEEK (-16336)>PEEK (-16336)
+PEEK (-16336): GOTO 85
75 FOR I=1 TO 6;X=PEEK (-16336)
80 NEXT I=W=0;
85 PLOT X,Y/3; COLOR=W: PLOT I,

K=Y+1;Y=J; GOTO 60
90 PRINT "INVALID. REENTER"
95 INPUT "COLOR (8 TO 15)",A;
IF E(0 OR E)15 THEN 90: RETURN
100 IF M THEN V=ABS (Y); VLIN
K=254;K=254+1 AT 1; S=51;1-2-
9: VTAB 21; TAB 13: PRINT 5
105 G=PEEK (-16336)>PEEK (-16336)
)+PEEK (-16336)>PEEK (-16336)
)+PEEK (-16336)>PEEK (-16336)
+PEEK (-16336)>PEEK (-16336)
60 IF S>=20 THEN 90
110 PRINT "CONGRATULATIONS, ";A#
"YOU WON": GOTO 165
120 PRINT "YOUR SCORE OF","S":15
"": GOTO 125+5(S/100)
125 PRINT "TERRIBLE": GOTO 165
130 PRINT "LOUSY."; GOTO 165
135 PRINT "POOR."; GOTO 165
140 PRINT "FAIR."; GOTO 165
145 PRINT "GOOD."; GOTO 165
150 PRINT "VERY GOOD."; GOTO 165
155 PRINT "EXCELLENT."; GOTO 165
160 PRINT "NEARLY PERFECT.";
165 PRINT "ANOTHER GAME ";A#: (Y/H)
": INPUT H#: IF H(1,1)="Y"
THEN 23: TEXT = CALL -
936: VTAB 10: TAB 10: PRINT "GAME OVER": END
170 Q=POW (0)20)/6: IF Q=0 THEN
0=0: IF Q>34 THEN 0=34: COLOR=
0: VLIN 0,0;5 AT 0: COLOR=0;
IF P=0 THEN 175: IF Q THEN
VLIN 0,0;1 AT 0:P=0: RETURN
175 IF P=0 THEN RETURN : IF G#34
THEN VLIN 0,6,39 AT 0;P=0;RETURN
180 FOR I=1 TO 80: J=PEEK (-16336)
): NEXT I: GOTO 50
COLOR DEMO PROGRAM LISTING

PROGRAM LISTING

10 DIM C(4); POKE 2,173; POKE 3,48; POKE 4,192; POKE 5,165
   POKE 6,8; POKE 7,32; POKE 8,160; POKE 9,252; POKE 10,165
   POKE 11,1; POKE 12,200

20 POKE 13,4; POKE 14,198; POKE 15,24; POKE 16,246; POKE 17
   ,5; POKE 18,199; POKE 19,1; POKE 20,76; POKE 21,2; POKE 22,0
   POKE 23,96

30 TEXT = CALL -936; VTABLE 4; TAB
   PRINT "4K COLOR DEMOS": PRINT
   PRINT "1 LINES": PRINT "2 CROSSES"
   PRINT "3 FELTING"

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

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

60 PRINT "HIT ANY KEY FOR NEW DEMO"
   :Z:0: PRINT : INPUT "WHICH DEMO"
   J: :I: :GR : IF J=0 AND I=11
   THEN GOTO 1004; GOTO 30

70 INPUT "WHICH DEMO WOULD YOU LIKE"
   :I: :GR : IF I AND I=12 THEN
   GOTO 1004; GOTO 30

100 I=I+1; MOD 79; J=I*(1339)+479
   :I-1: :GOSUB 2000; GOSUB 10000
   :GOTO 100

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

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

400 FOR J=1 TO 4; IF J=9: GOTO 16
   : NEXT J

410 FOR I=3 TO 1 STEP -1: CI(I)
   =CI(I); NEXT I: CI(I)= RND (16)
   : FOR I=1 TO 5: FOR J=1 TO 4

420 COLOR=C(J): L=J*5+i+1: X=39-
   L: HLIN K,L AT L; HLIN K,L AT
   L: HLIN K,L AT L: HLIN K,L AT
   K: NEXT J, I: GOSUB 10000; GOTO
   410

500 2=28: GOTO 900

600 COLOR=RHD (16); FOR I=0 TO
   18 STEP 2; J=39-J: HLIN K,J AT
   I: GOSUB 640; HLIN K,J AT I:
   GOSUB 640

610 HLIN I=2,J AT J: GOSUB 640
   : HLIN I=2,J AT I=2: GOSUB 640
   : NEXT I

620 COLOR=RHD (16); FOR I=18 TO 0
   STEP -2; J=39-J: HLIN I=2,
   J AT I+2: GOSUB 640; HLIN I+2,
   J AT J: GOSUB 640

630 HLIN I,J AT J: GOSUB 640; HLIN
   I,J AT I: GOSUB 640; NEXT I:
   GOSUB 10000; GOTO 600

640 K=I*7; L=K*K; K+K+L: X=32767
   /L<; PDL (8)<10>: POKE 0,K,X;
   POKE 1,L, MOD 256: POKE 24,
   L/256+I: CALL 2: RETURN

708 I= RND (28)+3; J=I*15: I*28+
   708; K=32767/<A; PDL (8)<10>
   : POKE 0,I; POKE 1,K MOD 256
   : POKE 24,(K*255)+1: CALL 2
   : GOSUB 10000; GOTO 700

808 X=3A:*188; P=R:L=28: W=4; Y=0
   : J=1; COLOR=6; HLIN 8,39 AT
   4: COLOR=9; GOSUB 888; COLOR=
   12: VLIN 5,5 AT X

810 N=W+F+P+R+W: COLOR=0; GOSUB
   888; VLIN 5,39 AT X; X=X+1: IF
   X<39 THEN 820: X=0: VLIN 5,39
   AT 1: VLIN 5,39 AT 2

820 F=A+R+Y+H/100: COLOR=12; GOSUB
   888; COLOR=9; VLIN 5,5 AT 2
   P: COLOR=15; PLOT X=2: FOR
   I=0 TO J: NEXT I: GOSUB 10000
   : GOTO 810

888 M=L+Y+1-L; I+1=H+1; VLIN L1,
   L2 AT X-1; VLIN L1_L2 AT X;
   VLIN L1_L2 AT X+1: RETURN

908 I=I+1 MOD 15: FOR V=0 TO 39
   : FOR X=0 TO 39: COLOR=1+X ABS
   (28-X)-Z)*ABS (28-Y)-Z)/25
   : PLOT X,Y: NEXT X,Y: GOSUB
   10000: GOTO 900

1000 CALL -936

1618 J=I+1 MOD 32: COLOR=J/2; VLIN
   0,39 AT 3+J: VTABLE 21+(J/2) MOD
   2; TAB 3+J; IF J MOD 2 THEN
   PRINT J/2: GOSUB 10000; GOTO
   1010

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

10000 IFpeek (-16384)*128 THEN RETURN
   : POKE -16388,0; POP : GOTO
   30
APPLE II STARDOK VERSION

This is a short description of how to play Stadrek 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:

1. # of stars in the ones place
2. # of bases in the tens place
3. # 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 (P) 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 'O' (Zero) and return. They are:
   1. Propulsion
   2. Regenerate
   3. Long range sensors
   4. Phasers
   5. Photon torpedoes
   6. Galaxy record
   7. Computer
   8. Pkope
   9. Shield energy
   10. Damage report
   11. Load photon torpedoes
2. The commands are invoked by typing the number referring 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 in quadrants the Enterprise will move.
   B. Course is Compass reading in degrees for the desired destination.
   C. A 2 regenerates the energy at the expense of time.
   D. A 3 gives the contents of the immediate adjacent quadrants.
   E. A 4 fires phasers at the expense of available energy.
   F. 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.
   G. A 6 and 10 all give information about the status of the ship and its environment.
   H. A 7 for information on loading and unloading of photon torpedoes at the expense of available energy.
   The answer should be a signed number. For example 15 or -2.

1. Enters a computer which will respond to the following instructions:
   1. Compute course
   2. Lock phasors
   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
1
2
3
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

AUTO num
Sets automatic line numbering mode. Starts at line number num and increments line numbers by 10. To exit AUTO mode, type a control X*, then type the letters "MAN" and press the return key.

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

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

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

DEL num1
Deletes line number num1.

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

DSP var
Sets debug mode that will display variable var every-time that it is changed along with the line number that caused the change. (NOTE: RUN command clears DSP mode so that DSP command is effective only if program is continued by a CON or GOTO command.)

HIMEM: expr
Sets highest memory location for use by BASIC at location specified by expression expr in decimal. HIMEM: may not be increased without destroying program. HIMEM: is automatically set at maximum RAM memory when BASIC is entered by a control B*.

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

GR
Sets mixed color graphics display mode. Clears screen to black. Resets scrolling window. Displays 40x40 squares in 15 colors on top of screen and 4 lines of text at bottom.

LIST
Lists entire program on screen.

LIST num1
Lists program line number num1.

LIST num1, num2
Lists program line number num1 through line number num2.
**LOAD** *expr.*  
Reads (Loads) a BASIC program from cassette tape.  
Start tape recorder before hitting return key. Two  
beeps and a ">" indicate a good load. "ERR" or "MEM"  
FULL ERR" message indicates a bad tape or poor recorder  
performance.

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

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

**NEW**  
Clears (Scratches) current BASIC program.

**NO DSP var**  
Clears DSP mode for variable *var*.

**NO TRACE**  
Clears TRACE mode.

**RUN**  
Clears variables to zero, undimensions all arrays and  
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 $X^C$.  

19
### 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 ( \text{ALPHA} = -(\text{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 \( N \* 2 \). |
| / | 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 \( XY4 = H-D \) | Subtruct |
| = | 130 \( \text{HEIGHT}=15 \) | Assignment operator; assigns a value to a variable. LET is optional |
| | 140 LET \( \text{SIZE}=7*5 \) | |
| | 150 \( A(8) = 2 \) | |
| | 155 \( \text{ALPHA}$ = "PLEASE" \) | |
## 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$(1,1)=&quot;Y&quot; THEN 500</td>
<td>String variable &quot;equals&quot; string variable.</td>
</tr>
<tr>
<td># or &lt; &gt;</td>
<td>180 IF ALPHA #X*Y THEN 500</td>
<td>Expression &quot;does not equal&quot; expression.</td>
</tr>
<tr>
<td>#</td>
<td>190 IF A$ # &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 500</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>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(expr)</td>
<td>Gives absolute value of the expression expr.</td>
</tr>
<tr>
<td>ASC(str$)</td>
<td>Gives decimal ASCII value of designated string variable str$. 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$)</td>
<td>Gives current length of designated string variable str$; i.e., number of characters.</td>
</tr>
<tr>
<td>PDL(expr)</td>
<td>Gives number between 0 and 255 corresponding to paddle position on game paddle number designated by expression expr and must be legal paddle (0, 1, 2, or 3) or else 255 is returned.</td>
</tr>
<tr>
<td>PEEK(expr)</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 FFF0 is -16.</td>
</tr>
<tr>
<td>RND(expr)</td>
<td>Gives random number between 0 and (expression expr -1) if expression expr is positive; if minus, it gives random number between 0 and (expression expr +1).</td>
</tr>
<tr>
<td>SCRNS(expr1, expr2)</td>
<td>Gives color (number between 0 and 15) of screen at horizontal location designated by expression expr1 and vertical location designated by expression expr2. Range of expression expr1 is 0 to 39. Range of expression expr2 is 0 to 39 if in standard mixed color graphics display mode as set by GR command or 0 to 47 if in all color mode set by POKE -16304,0: POKE -16302,0.</td>
</tr>
<tr>
<td>SGN(expr)</td>
<td>Gives sign (not sine) of expression expr i.e., -1 if expression expr is negative, zero if 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 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 variable 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 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 var1 (expr1)
str$ (expr2)
var2 (expr3)

50 DIM A(20), B(10)
60 DIM B$(30)
70 DIM C (2)

Illegal:

80 DIM A(30)
Legal:

85 DIM C(1000)

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

DSP var

Legal:

90 DSP AX; DSP L

Illegal:

100 DSP AX,B
102 DSP 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 var=</td>
<td>110 FOR L=0 to 39</td>
<td>Begins FOR...NEXT loop, initializes variable var to value of expression expr1 then increments it by amount in expression expr2 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>expr1 TO expr2</td>
<td>120 FOR X=Y1 TO Y3</td>
<td></td>
</tr>
<tr>
<td>STEP expr3</td>
<td>130 FOR I=39 TO 1</td>
<td></td>
</tr>
<tr>
<td>150 GOSUB 100 *J2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GOSUB expr</td>
<td>140 GOSUB 500</td>
<td>Causes branch to BASIC subroutine starting at legal line number specified by expression expr. Subroutines may be nested up to 16 levels.</td>
</tr>
<tr>
<td>GOTO expr</td>
<td>160 GOTO 200</td>
<td>Causes immediate jump to legal line number specified by expression expr.</td>
</tr>
<tr>
<td>170 GOTO ALPHA+100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GR</td>
<td>180 GR</td>
<td>Sets mixed standard resolution color graphics mode. Initializes COLOR = 0 (Black) for top 40x40 of screen and sets scrolling window to lines 21 through 24 by 40 characters for four lines of text at bottom of screen. Example 190 sets all color mode (40x48 field) with no text at bottom of screen.</td>
</tr>
<tr>
<td>190 GR: POKE -16302,0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HLIN expr1,</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>expr2 AT expr3</td>
<td>210 HLIN Z,Z+6 AT I</td>
<td></td>
</tr>
<tr>
<td>Note:</td>
<td>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>
<td></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 260 is illegal but may be implemented as shown in example 270.

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

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

Assignment operator. "LET" is optional

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

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

Turns-off DSP debug mode for variable

Turns-off TRACE debug mode
PLOT, expr1, expr2 400 PLOT 15, 25
        400 PLT XV,YV

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

POKE expr1, expr2 420 POKE 20, 40
          430 POKE 7*256, XMOD255

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

POP 440 POP

"POPS" nested GOSUB return stack address by one.

PRINT var1, var, str$ 450 PRINT L1
          460 PRINT L1, X2
          470 PRINT "AMT=";DX
          480 PRINT A$;B$;
          490 PRINT
          492 PRINT "HELLO"
          494 PRINT 2+3

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

PR# expr 500 PR# 7

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

REM 510 REM REMARK

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

RETURN 520 RETURN
          530 IF X= 5 THEN
        RETURN

Causes branch to statement following last GOSUB; i.e., RETURN ends a subroutine. Do not confuse "RETURN" statement with Return key on keyboard.
**TAB expr**
530  TAB 24
540  TAB I+24
550  IF A#B THEN
     TAB 20

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

**TEXT**
550  TEXT
560  TEXT: CALL-936

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

**TRACE**
570  TRACE
580  IFN > 32000
     THEN  TRACE

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

**VLIN expr1, expr2**
590  VLIN 0, 39AT15
   AT  expr3
600  VLIN Z,Z+6ATY

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

**VTAB expr**
610  VTAB 18
620  VTAB Z+2

Similar to TAB. Moves cursor to absolute vertical position specified by expression expr in the range 1 to 24. VTAB 1 is top line on screen; VTAB24 is bottom.
SPECIAL CONTROL AND EDITING CHARACTERS

"Control" characters are indicated by a super-scripted "C" such as G\(^C\). They are obtained by holding down the CTRL key while typing the 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 transfered 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;&lt;&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;-&gt;&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>

Table A: APPLE II COLORS AS SET BY COLOR =

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

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

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

### Miscellaneous

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

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

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**: Monitor and BASIC Routines in ROM
- **E000**: Future enhancement or user supplied PROMS
- **D000**: Peripheral I/O
- **C000**: User specified RAM memory size
- **XX**: User Workspace
- **7FF**: Screen Memory
- **400**: Internal Workspace
READ/SAVE DATA SUBROUTINE

INTRODUCTION

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

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

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

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

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

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

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

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

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

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

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

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

DATA - ASCII equivalents with high order bit set.

STRING TERMINATOR (ST) - none high order bit set character indicating END of string.

There are two parts of any BASIC program represented in memory. One is the location of the variables used for the program, and the other is the actual BASIC program statements. As it turns out, the mapping of these within memory is a straightforward process. Program statements are placed into memory starting at the top of RAM memory* unless manually shifted by the "HIMEM:" command, and are pushed down as each new (numerically larger) line numbered statement is entered into the system. Figure 1a illustrates this process diagramatically. Variables on the other hand are mapped into memory starting at the lowest position of RAM memory - hex $800$ (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 LOMEN and HIMEN: But this is expected because upon using the BC command both memory pointers are initialized indicating no program statements and no variables.

To illustrate what a variable table looks like in Apple II memory suppose we want to assign the numeric variable A ($C1$ is the ASCII equivalent of a with the high order bit set) the value of -1 (FF FF in hex) and then examine the memory contents. The steps in this process are outlined in example I. Variable A is defined as equal to -1 (step 1). Then for convenience another variable - B - is defined as equal to 0 (step 2). Now that the variable table has been defined use of statement 2a indicates that CM is equal to 2060 (step 3). LOMEN has not been readjusted so it is equal to 2048. Therefore the variable table resides in memory from 2048 ($800$ hex) to 2060 ($80C$). Depressing the "RESET" key places the Apple II into the monitor mode (step 4).

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

READ/SAVE ROUTINE

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

The variables used in this subroutine are defined as follows:

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

SAVING A DATA TABLE

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

READING A DATA TABLE

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

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

CONCLUSION

Reading and Saving data in the format of a variable table is a relatively 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 \rightarrow PP
b) PRINT PEEK(202) + PEEK(203)*256 \rightarrow CM

Figure 2

Figure 3

$800.80C$ rewritten with labelling
READ/SAVE PROGRAM

0     A=0

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

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

26    PRINT "DATA TABLE SAVED": RETURN

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

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    PRINT "DATA READ IN": RETURN

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

NOTE: CM, LM and A must be defined within the main program.

This must be the first statement in the program. It is initially 0, but if data is to be saved, it will equal the length of the data base.

This statement moves command to the main program.

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

Writing data table to tape

Returning control to main program.

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

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

Returning control to main program.
1  >A=1
   >

2  >B=0
   >

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

4  >
   *

5  *800.80C

Computer responds with:
0800- C1 00 86 08 FF FF C2 00
0808  0C 08 00 00 00

Example 1

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 2

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 RECORDER
MODE": INPUT "AND HIT RETURN"
,A#

130 CALL -996: PRINT "NOW WRITING DA
TA TO TAPE": GOSUB 20

135 PRINT "NOW THE DATA IS SAVED"

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)=0: PRINT
"X(";I;")= ";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 "ALL THE DATA READ IN"

190 FOR I=1 TO 20: PRINT "X";I;
"=";X(I): NEXT I

195 PRINT "THIS IS THE END"

200 END
INTRODUCTION

Computers can perform marvelous feats of mathematical computation at well beyond the speed capable of most human minds. They are fast, cold and accurate; man on the other hand is slower, has emotion, and makes errors. These differences create problems when the two interact with one another. So to reduce this problem humanizing of the computer is needed. Humanizing means incorporating within the computer procedures that aid in a program's usage. One such technique is the addition of a tone subroutine. This paper discusses the incorporation and usage of a tone subroutine within the Apple II computer.

Tone Generation

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

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

Machine Language Program

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

Basic Program

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

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

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

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

CONCLUSION

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

: POKE 7, 4: POKE 8, 198: POKE 9, 1: POKE 10, 240
32005 POKE 11, 8: POKE 12, 202: POKE 13, 208: POKE 14, 246: POKE 15
, 166: POKE 16, 0: POKE 17, 76
: POKE 18, 2: POKE 19, 0: POKE 20, 96: RETURN

FIGURE 2. BASIC "POKES"

25 POKE $8,P: POKE 1,0: CALL 2:
RETURN

FIGURE 3. GOSUB
High-Resolution Operating Subroutines

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

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

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

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

**INIT** Initializes High-Resolution Graphics mode.
From BASIC: CALL 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 3086 (or CALL -12274)
From machine language: JSR $C0E (or JSR $D00E)

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

**PLOT** Plots a point on the screen.
From BASIC: CALL 3780 (or CALL -11580)
From machine language: JSR $C7C (or JSR $D07C)

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

PLOT (continued)
(top of screen) to 159 (bottom of screen) and is passed in
location 802 or the A-register; but the X (horizontal) coordinate
can range from $\theta$ (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 $X$ ($X \text{ LO}$)
and $Y$ ($X \text{ HI}$). The color of the point to be plotted must be set
in location 812 ($32C$). Four colors are possible: $\theta$ 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 $D026$)

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.
LINE Draws a line on the screen.

From BASIC: CALL 3786 (or CALL -11574)

From machine language: JSR $C95 (or JSR $D95)

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 806 from BASIC or $32F from machine language. The rotation factor specifies one of 64 possible angles of rotation for the shape. A rotation factor of 0 will cause the shape to be drawn right-side up, where a rotation factor of 16 will draw the shape rotated 90° clockwise, etc. The rotation factor is passed in location 807 from BASIC or in the A-register from machine language.

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

The three sections are arranged in a byte like this:

```
0 0 5 4 3 2 1 0
```

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

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

Suppose we want to draw a shape like this:

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

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

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

Now draw a table like the one in Figure 1. For each vector in the line, figure the bit code and place it in the next available section in the table. If it will not fit or is a $\emptyset \emptyset$ at the end of a byte, then skip that section and go on to the next. When you have finished
SHAPE (continued)

coding all vectors, check your work to make sure it is accurate. Then make another table (as in figure 2) and re-copy the coded vectors from the first table. Then decode the vector information into a series of hexadecimal bytes, using the hexadecimal code table in figure 3. This series of hexadecimal bytes is your shape definition table, which you can now put into the Apple II's memory and use to draw that shape on the screen.
Shape vectors:

```
C B A
∅ 01 01 01
1 00 10 10
2 10 00 00
3 10 10 10
4 00 11 10
5 11 11 10
6 00 00 10
7 00 00 00
8 00 00 00
```

Figure 1.

Codes:
```
<table>
<thead>
<tr>
<th>up</th>
</tr>
</thead>
<tbody>
<tr>
<td>up</td>
</tr>
<tr>
<td>up</td>
</tr>
<tr>
<td>up</td>
</tr>
<tr>
<td>left</td>
</tr>
<tr>
<td>right</td>
</tr>
<tr>
<td>left</td>
</tr>
<tr>
<td>right</td>
</tr>
<tr>
<td>left</td>
</tr>
<tr>
<td>right</td>
</tr>
<tr>
<td>left</td>
</tr>
<tr>
<td>right</td>
</tr>
</tbody>
</table>

Empty:
```

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

Figure 2.

\[ C | \begin{array}{|c|c|} \hline B & A \hline \hline ∅ & 0000 \quad 0010 \quad 0011 \quad 0010 \quad 0001 \quad 0000 \hline 1 & 0011 \quad 1111 \quad 1111 \quad 1111 \quad 1111 \quad 1111 \hline 2 & 0110 \quad 0000 \quad 0001 \quad 0010 \quad 0100 \quad 0101 \hline 3 & 0110 \quad 0000 \quad 0001 \quad 0010 \quad 0100 \quad 0101 \hline 4 & 0110 \quad 0000 \quad 0001 \quad 0010 \quad 0100 \quad 0101 \hline 5 & 0110 \quad 0000 \quad 0001 \quad 0010 \quad 0100 \quad 0101 \hline 6 & 0110 \quad 0000 \quad 0001 \quad 0010 \quad 0100 \quad 0101 \hline 7 & 0110 \quad 0000 \quad 0001 \quad 0010 \quad 0100 \quad 0101 \hline 8 & 0110 \quad 0000 \quad 0001 \quad 0010 \quad 0100 \quad 0101 \hline 9 & 0110 \quad 0000 \quad 0001 \quad 0010 \quad 0100 \quad 0101 \hline \hline \end{array}\right] = \begin{array}{|c|c|} \hline 1 & 12 \hline 2 & 3F \hline 3 & 20 \hline 4 & 64 \hline 5 & 2D \hline 6 & 15 \hline 7 & 36 \hline 8 & 1E \hline 9 & 07 \hline \hline \end{array}
```

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

Figure 2.
ROD'S COLOR PATTERN

PROGRAM DESCRIPTION

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

REQUIREMENTS

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

PROGRAM LISTING

100 GR
105 FOR W=3 TO 50
110 FOR I=1 TO 19
115 FOR J=0 TO 19
120 X=I+J
125 COLOR=I+(I+J)*J+J/12
130 PLOT I,K; PLOT K,I; PLOT 40
   -I,40-K
135 PLOT 40-K,40-I; PLOT K,40-I;
   PLOT 40-I,K; PLOT 1,40-K; PLOT
   40-K,1
140 NEXT J,1
145 NEXT W: GOTO 105
5 REM PONG BY WENDELL BITTER
10 REM 7/7/77
15 REM PADDLE SWITCHES CONTROL
PADDLE SIZE AFTER A MISS
OR DURING A HIT
20 GR
25 DIM P(3); DIM HP(18)
30 A=30:B=11:C=-1
35 COLOR=13: HLINE 1,38 AT 0: HLINE 1,30 AT 39
40 CALL -936: VTRB 23: INPUT "HAND OR PONG? ",HP
45 INPUT "PADDLE SIZE (1-6) ", PS: IF PS<1 OR PS>6 THEN 45 
:5=PS-1
50 CALL -936
55 IF HP(X:1)="H" THEN 205
60 H=1: COLOR=13: VLINE 0,39 AT 39: GOTO 205
65 FOR X=0 TO 1: Y=33+6: IF Y>11 THEN 65 
70 Y=Y+Y: IF Y11 THEN Y=11: IF Y<10 
:70 THEN 70
75 V=V+1: FOR T=1 TO 5: H=H-1: V= PEEK (-16336); NEXT T
80 IF X=0 OR X=39 THEN GO: COLOR= 0: PLOT X-C,Y: COLOR=15: PLOT 
X,Y
85 Y=Y: IF X MDO 2=0 THEN 60 GOSUB 235: NEXT X
90 GOSUB 235
95 IF SORW(X,Y+Y+Y+Y+Y+Y+6):0 THEN 165
100 FOR T=1 TO 10: W=PEEK (-16336); NEXT T
105 IF H RAND 0: GO: 130
110 PP=P(2:36)
115 IF Y<PP THEN V=3: IF Y<PP+1 
:THEN V=2: IF Y<PP+2 THEN V= 
1
120 IF Y<PP+3 THEN V=-1: IF Y<PP+ 
4 THEN V=-2: IF Y<PP+5 THEN 
V=-3
125 IF S=0 THEN V=5- RND (7)
130 COLOR=0: PLOT X-C,Y
135 IF (X AND CY) OR (YV= ABS 
(Y) AND X=0) THEN V=4- RND 
(9)
140 IF X=0 THEN YV= ABS (Y)
145 A=39-A: B=39-B: C=-C
150 IF PEEK (-16286)>127 AND 5# 
5 THEN 5+5+1
155 IF PEEK (-16287)>127 AND 5# 
0 THEN 5-5+1
160 GOTO 65
165 COLOR=0: PLOT X-C,Y
170 COLOR=15: PLOT X,Y+V(Y+VV- 
1 AND Y+Y4(40)
175 FOR T=1 TO 75: H=PEEK (-16336) 
+ PEEK (-16336): PEEK (-16336); 
NEXT T
180 IF X=0 THEN SR=5+1: IF X=39 
: THEN SL=SL+1
185 VTRB 23: TAB 7: PRINT SL;: TAB 
3: PRINT 5R
190 COLOR=0: PLOT X-C,Y
195 IF SL=1 OR SR=1 THEN 260
200 COLOR=0: PLOT X,Y+Y(Y+Y)-1 
AND Y+Y4(40)
205 FOR T=1 TO 75: IF T MOD 5 #0 
: THEN 210: IF PEEK (-16286) 
+127 AND 5$ THEN 5$+1: IF 
PEEK (-16287)>127 AND 5# THEN 
5$-1
210 GOSUB 235: NEXT T
215 Y=Y-P(3): IF X=0 THEN Y=Y-P(1) 
:220 IF H THEN YY= RAND (37)+1
225 V=-1- RAND (3)
230 GOTO 65
235 IF H THEN 245: P(1)=C P( 
1)-24:28)/115: IF P(1)<P(3) 
: THEN 245: IF P(1)<P(3) THEN 
P(1)=0: IF P(1)+539 THEN P(1) 
=39-5
240 COLOR=6: VLINE P(1),P(1)+5 AT 
39: COLOR=0: IF P(1)<P(3) THEN 
VLINE 0,P(1)-1 AT 39: IF P(1) 
>P(3) THEN VLINE P(1)+5+1,39 
AT 39: P(3)=P(1)
245 P(0)=-C P(0)-24:28)/115 
: IF P(0)<O THEN P(0)=0: IF 
P(0)<P(2) THEN RETURN: IF 
P(0)+5<39 THEN P(0)=39-5
250 COLOR=6: VLINE P(0),P(0)+5 AT 
0: COLOR=0: IF P(0)<P(2) THEN 
VLINE 0,P(0)-1 AT 0: IF P(0) 
P(2) THEN VLINE P(0)+5+1,39 
AT 0: P(2)=P(0): RETURN
255 PRINT "": END
260 PRINT "": END
265 END
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 6,0
  : POKE 7,32: POKE 8,168: POKE 9,252: POKE 10,165: POKE 11,1: POKE 12,208: POKE 13,4
16 POKE 14,198: POKE 15,24: POKE 16,240: POKE 17,5: POKE 18,188: POKE 19,1: POKE 20,76:
  : POKE 21,2: POKE 22,0: POKE 23,96
15 DIM B(40): TEXT: CALL -936: GOTO 90
20 CALL -936: GOTO 90
30 B$="***************
  *********": RETURN
35 B$="COLOR SKETCH": RETURN
40 B$="COPYRIGHT APPLE COMPUTER 197": RETURN
45 B$="THIS PROGRAM ALLOWS YOU TO "
  : RETURN
50 B$="SKETCH COLORED FIGURES IN" 
  : RETURN
55 B$="LOW RESOLUTION GRAPHICS WITH " 
  : RETURN
60 KK=20:TON=20:GOSUB 85: RETURN
65 KK=10:TON=10:GOSUB 85: RETURN
70 KK=20:TON=50:GOSUB 85:KK=30:TON=90:GOSUB 85: RETURN
75 KK=20:TON=20:GOSUB 85: RETURN
85 POKE 1,TON MOD 256: POKE 24
  ,TON/256+1: POKE 0, KK: CALL 2: RETURN
90 GOSUB 39: GOSUB 25: PRINT 
  : TAB 19: GOSUB 35: GOSUB 25 
  : PRINT : GOSUB 38: GOSUB 25 
100 PRINT : PRINT: GOSUB 70: INPUT
  
  WHEN READY HIT RETURN",B$ 
105 GR
110 B$="ABCDEFGHIJKLMNOPQRSTUVWXYZ": CALL -936
115 FOR Z=0 TO 15: COLOR=Z: PLOT 
  Z+2+4,39: YTAB 21: GOSUB 75
  : TAB Z+2+4: PRINT B$(Z+1,Z+=1): GOSUB 75: NEXT Z: TAB 1
120 YTAB 22:B$="TYPE A LETTER TO CHANGE COLOR.": GOSUB 25: PRINT
  :B$="TYPE SPACE BAR TO STOP PLOT 
  .": GOSUB 25: PRINT
125 Y= PDL (1)*30/255:X= PDL (0 )
  *39/255: YTAB 24: TAB 1: PRINT
  "CURSOR POSITION: X=",X," Y=" 
  Y=" :;
130 IF PEEK (-16364)>127 THEN 145
  : IF X=X AND Y=Y THEN 125
  : COLOR=C: PLOT X,Y: IF
  NOT FLAG THEN 135: COLOR=C: PLOT X,Y
135 C2=SCR(X,Y):C3=15: IF C2=
  15 THEN C3=5: COLOR=C3: PLOT 
  X,Y:XI=X,Y1=Y 
140 GOTO 125
145 IF PEEK (-16384)#160 THEN 155
  :FLAG=0: POKE -16368,0: POKE 
  34,20: COLOR=9: HBH X,39 AT 
  39: CALL -936
150 PRINT :B$="CONTINUE OR STOP"
  : YTAB 24: GOSUB 25: INPUT
  " (C/S )",B$: IF B$(I,1)="C"
  THEN 110: PRINT "END": END
155 FLAG=1:C= PEEK (-16384)-193
  : POKE -16368,0: 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 0-25-77
0 Woz (Apple Computer)
10 DIM R(6),C(6),X(6),K(6) AS INTEGER
20 DIM X(1)=2;X(2)=12;X(3)=1;X(4)=13;X(5)=3;X(6)=9;X(7)=15
30 DIM D(5)=5:X=X;"GRYDOWK"
40 TEXT 1; CALL -936; PRINT "

WELCOME

TO THE GAME OF MASTERMIND!

YOUR OBJECT IS TO GUESS 5 COLOR
5 WHICH:
60 PRINT "I WILL MAKE UP" IN THE MINIMUM NUMBER OF GUESSES. THERE
GUESS ARE EIGHT DIFFERENT COLORS TO
70 PRINT "CHOOSE FROM."
80 PRINT "

FEWER THAN 7 GUESSES--EXC
ERROR: PRINT "7 TO 9 GUESSES"
90 PRINT "TO 10 GUESSES--AVERAGE"
100 PRINT "MORE THAN 14 GUESSES--POOR"

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

300 CALL -300: IF PEEK (-16364) <132 THEN 100: POKE -16368,
310 0: GR: PRINT FOR I=1 TO
320 0:C(I)=END (8)*1: COLOR=X(I)
330 L: HLN L=I+4,1+4 AT 39: PRINT
340 ;";X(I),I); NEXT I
350 TRY=0: PRINT "LETTER
360 KEYS FOR COLOR CHANGE": PRINT
370 "ARROW KEYS FOR ADVANCE AND BACK": PRINT "HIT RETURN TO ACCEPT GUESS #"

380 Y=TRY*2 MOD 36+1:TRY=TRY+1:
390 TAB 32: PRINT TRY: COLOR=
400 H L=I+29 AT Y:FLASH=11 FOR
410 N=1 TO 5:R(N)=8: GOSUB 1000
420 NEXT N=N+1
430 FOR WAIT=1 TO 10:KEY= PEEK
440 (-16364): IF KEY<132 THEN 310
450 : POKE -16368,0:FLASH=11 FOR
460 I=1 TO 3: IF KEY(X:ASC(X(I))
470 ) THEN NEXT 1: IF I=9 THEN
480 310:R(N)=1: KEY=149
490 GOSUB 1000: IF KEY=141 THEN
500 : IF KEY=136 AND N=1 OR
510 KEY=149 AND N<6 THEN N=N+KEY;5-29: NEXT WAIT:FLASH=1-FLASH:
520 GOTO 300
530 COLOR=X(1)=0: FOR I=1 TO 5;
540 D(I)=C(I)=I=1: GOSUB 2000: NEXT
550 I: IF N=5 THEN 500: COLOR=5
560 FOR J=1 TO 5: FOR I=1 TO
570 GOSUB 2000: NEXT 1,I; GOTO
580 280
590 PRINT: PRINT "YOU GOT IT IN"
600 ,TRY,"TRIES (":; IF TRY<7 THEN
610 PRINT "EXCELLENT":; IF TRY>6 AND TRY<10 THEN PRINT "GOOD"
620 ;
630 510 IF TRY<9 AND TRY<15 THEN PRINT
640 "AVERAGE":; IF TRY>14 THEN
650 PRINT "POOR":; PRINT "")": CALL
660 -304: TAB 5: PRINT "HIT ANY KEY
670 TO PLAY AGAIN": GOTO 100
680 IF N<6 THEN RETURN: COLOR=
690 X(R(N))=FLASH: HLN N=4-2,N+4 AT Y: RETURN
700 IF X(R(I))=X(J) THEN RETURN
710 N=N+1: PLOT 21+M+H,Y: PRINT
720 "R(I)=0:X(J)=9: RETURN
73000 REM CALL -304 SETS INVERSE VID
74010 REM CALL -300 SETS NORMAL VID
75020 REM PEEK(-16364) IS KB0 (ASCII)
760 FOR I=127 THEN STROBE SET
77030 REM POKE-16360 CLRS KB0 STROBE
78040 REM CALL-936 CLEAR SCREEN AND
790 TAB CURSOR TO TOP LEFT.
80050 REM IN 310, KEY<5-29=-1 OR +1
810 (ARROW KEY=136 OR 149 ASCII)
82000 REM STATS 10-50 INTRO
83010 REM STATS 100-110 NEW SETUP
84020 REM STMT 200 NEW GUESS
85030 REM STMT 300-310 USER INPUT
86040 REM STMT 400 GUESS EVAL
87050 REM STMT 500-510 WIN
88060 REM SUBR 1000 COLOR LINE
89070 REM SUBR 2000 MATCH TEST
90080 REM SUBR 3000 ALIAS
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


15 GOTO 65
20 TT=3: GOSUB 38: RETURN
25 PRINT "**"; RETURN
30 XX=8:TON=500: GOSUB 45: RETURN
35 XX=9:TON=258: GOSUB 45: RETURN
40 XX=9:TON=258: GOSUB 45:XX=9: TON=258: GOSUB 45: RETURN
45 POKE 1,TON MOD 256: POKE 24,TON/256+1: POKE 0,XX: CALL 2: RETURN
50 A=19-(P*B(I)/100):(P=100: C(I)=P=100*G(I): G(I)=P=100*C(I))/100:B(I)=G(I)/100: RETURN
55 A=6*(P=100)*3*(C(I))/100:B(I)=P=100*6*(C(I))/100):
RETURN
60 XX=8:TM=500: GOSUB 78:XX=9:
TM=506: GOSUB 78: RETURN
65 XX=7:TM=18: GOSUB 78: RETURN

70 POKE 1,1: TM MOD 256: POKE 24,TM/256+1: POKE 0,XX: CALL 2: RETURN
75 GOSUB 68: INPUT "DATE (M,D,Y) " : RETURN
80 A=Y-(K)3:N=Y MOD 58*365-Y
56*A+R/4-R/100*N-31*N-12*N
7-M/5-3*(M)/2): IF N=0 THEN N=H-21252: RETURN
85 DIM H(10),B(3),C(3), BV(3):B(1)=368:B(2)=286:B(3)
=242:C(1)=575:C(2)=708:C(3)
=B2:BV(1)=23:BV(2)=28
90 BV(3)=33: TEXT : CALL -936:
POKE 34,208: GOSUB 28: GOSUB 25:
PRINT "APPLE II BIORHYTHM (4K)"
: PRINT "1977 APPLE COMPUTER INC."
98 GOSUB 25: TAB 5: PRINT "COPYRIGHT": RETURN
100 GOSUB 68: INPUT "NAME ",N#: Y:
VTAB 22: PRINT NA#: VTAB 24:
: PRINT "BIRTH ":: GOSUB 75
105 PRINT "FORECAST ":: GOSUB 75
: N=N-H: IF N=0 THEN N=H-21252:
: VTAB 23: TAB 10: PRINT "FORECAST DATE ";; Y: VTAB 24:
: CALL -968
110 J=1: GR : POKE 34,23: FOR X=18 TO 28: COLOR=3: HLIN 0,31
AT X: NEXT X: HLIN 1,3 AT 3: HLIN 1,3 AT 37: VLIN 2,4
AT 2: VTAB 21
115 FOR Y=1 TO 31 STEP 3: PRINT "; IF Y(10 THEN PRINT ";
: PRINT ";; NEXT Y: PRINT ": P E": VTAB 24
120 VTAB 23: PRINT "DAYS LIVED ":
: N: FOR I=1 TO 3: COLOR=I(
I+1)+6*(I=2)+0*(I=3): VLIN 0,39 AT 33+I: VTAB 24
125 FOR X=0 TO 31:P=N MOD BV(I)
130 PRINT : INPUT "ANOTHER PLOT (Y/N) " ;: IF B$(1,1)="Y" THEN
90: END

62
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

1 TEXT : CALL -936
2 PRINT "WELCOME TO THE DRAGON'S MAZE"
3 PRINT "YOU MAY WATCH WHILE I BUILD A MAZE,"
4 PRINT "BUT WHEN IT'S COMPLETE, I'LL ERASE"
5 PRINT "THE PICTURE. THEN YOU'LL ONLY SEE THE WALLS AS YOU BUMP INTO THEM."
6 PRINT "TO MOVE, YOU HIT 'R' FOR RIGHT,"
7 PRINT "L" FOR LEFT, 'U' FOR UP, AND "":
8 PRINT "D" FOR DOWN. DO NOT HIT RETURN!"
9 PRINT
10 PRINT "THE OBJECT IS FOR YOU (THE GREEN DOT)"
11 PRINT "TO GET TO THE DOOR ON THE RIGHT SIDE"
12 PRINT "BEFORE THE DRAGON (THE RED DOT) EATS".
13 PRINT "YOU."
14 PRINT "BEWARE!!!!!!!!! SOMETIMES THE DRAGON"
15 PRINT "GETS REAL MAD, AND CLIMBS OVER A WALL."
16 PRINT "BUT MOST OF THE TIME, HE CAN'T GO OVER"
17 PRINT "AND HAS TO GO AROUND."
18 PRINT
19 PRINT "HINT: YOU CAN OFTEN TELL WHERE A WALL"
20 PRINT "IS, EVEN BEFORE YOU CAN SEE IT, BY"
21 PRINT "THE FACT THAT THE DRAGON CAN'T GET"
22 PRINT "THROUGH IT!"
23 PRINT
24 DIM MK(3)
25 PRINT 'TYPE 'GO' TO BEGIN '
:: INPUT A#
26 GR = COLOR=15
27 CALL -936: PRINT "DRAGON MAZE"
:: TAB (25): PRINT "GARY J. SHAH"
28 NL FOR I=0 TO 39 STEP 3: VLIN 0,39 AT I: HLIN 0,39 AT I: NEXT I
29 COLOR=8
30 S=1600
31 K=169
32 PRINT FOR I=1 TO 169: K(I)=0: NEXT I
33 PRINT FOR I=1+169: K(I)=1: NEXT I
34 PRINT X= RND (13)+1: Y= RND (13)+1 : C=169
35 IF C=1 THEN 1200
36 R=80:D=80:L=80:U=80:K=8+13*(Y-1)
):MK)=ABS (MK)+C=C-1
37 IF X=13 THEN 1060:R=MK+1))
38 IF Y=13 THEN 1070:D=MK+1))
39 IF X=1 THEN 1080:L=MK-1))
40 IF Y=1 THEN 1090:U=MK-1))
41 IF R=0+D=0+L=0+U=0 THEN 1170
42 IF (0=3 AND RND (10)*2 OR =0 THEN 1170
43 =0 D= RND (4)
44 GOTO 1120
45 IF NOT R THEN 1110: MK=MK
46 +1: X=X+1
47 VLIN 3*Y-2,3*Y-1 AT 3*Y(X-1)
48 GOTO 1135
49 GOTO 1136
50 IF NOT D THEN 1110: MK=MK
51 +1:Y=Y+1
52 HLIN 3*Y-2,3*Y-1 AT 3*Y(Y-1)
53 GOTO 1145
54 GOTO 1146
55 IF NOT L THEN 1110: MK=MK
56 -1:Y=Y-1
57 VLIN 3*Y-2,3*Y-1 AT 3*Y+X
58 GOTO 1155
59 VLIN 3*Y-2,3*Y-1 AT 3*Y+X
60 GOTO 1156
61 IF NOT U THEN 1110: MK=MK
62 -1: Y=Y-1
63 HLIN 3*Y-2,3*Y-1 AT 3*Y: GOTO
64 GOTO 1165
65 IF NOT H THEN 1110: MK=MK
66 -1: X=X-1
67 HLIN 3*Y-2,3*Y-1 AT 3*Y: GOTO
68 GOTO 1170
69 IF MK*X=13*(Y-1)):@ THEN 1170
70 IF MK*X=13*(Y-1)):@ GOTO 1190
71 C=C+1: GOTO 1200
72 GOSUB 5000: PRINT "THE MAZE IS READY!"
73 GOTO 1205
74 GR = COLOR=15
75 VLIN 0,39 AT 0: VLIN 0,39 AT 39: HLIN 0,39 AT 0: HLIN 0, 39 AT 39
76 GOTO 1220
77 X=1: Y= RND (13)+1: COLOR=8: PLOT 3*Y-2,3*Y-2

64
DRAGON MAZE cont.

7110 DX=-1:DY=0: GOTO 7020
7150 IF SY=1 THEN 7005: IF T<(SX+13*(SY-1))/9 THEN 7160: IF
8000 GOSUB 5000: GOSUB 5000: GOSUB 5000: PRINT "THE DRA
9999 END
APPLE II Firmware

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

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

<table>
<thead>
<tr>
<th>Command Format</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examine Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrs</td>
<td>*C0F2</td>
<td>Examines (displays) single memory location of (adrs)</td>
</tr>
<tr>
<td>adrs1.adrs2</td>
<td>*1024.1048</td>
<td>Examines (displays) range of memory from (adrs1) thru (adrs2)</td>
</tr>
<tr>
<td>(return)</td>
<td>* (return)</td>
<td>Examines (displays) next 8 memory locations.</td>
</tr>
<tr>
<td>.adrs2</td>
<td>*.4096</td>
<td>Examines (displays) memory from current location through location (adrs2)</td>
</tr>
<tr>
<td>Change Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrs:data</td>
<td>*A256:EF 20 43</td>
<td>Deposits data into memory starting at location (adrs).</td>
</tr>
<tr>
<td>:data data</td>
<td>*F0 A2 12</td>
<td>Deposits data into memory starting after (adrs) last used for deposits.</td>
</tr>
<tr>
<td>Move Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrs1&lt;adrs2.</td>
<td>*100&lt;B010.B410M</td>
<td>Copy the data now in the memory range from (adrs2) to (adrs3) into memory locations starting at (adrs1).</td>
</tr>
<tr>
<td>adrs3M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verify Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrs1&lt;adrs2.</td>
<td>*100&lt;B010.B410V</td>
<td>Verify that block of data in memory range from (adrs2) to (adrs3) exactly matches data block starting at memory location (adrs1) and displays differences if any.</td>
</tr>
<tr>
<td>adrs3V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command Format</td>
<td>Example</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Cassette I/O</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adrsl.adrs2R</td>
<td>*300.4FFR</td>
<td>Reads cassette data into specified memory (adrs) range. Record length must be same as memory range or an error will occur.</td>
</tr>
<tr>
<td></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>N</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></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>adrsl (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) (6502</td>
<td>STA $1FF</td>
<td>Assembles instruction into next available memory location. (Note space between &quot;!&quot;) and instruction</td>
</tr>
<tr>
<td>mnemonic instruction)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(TURN-OFF)</td>
<td>!(Reset Button)</td>
<td>Exits mini-assembler and returns to system monitor.</td>
</tr>
</tbody>
</table>

### Monitor Program Execution and Debugging

- **adrsG** *300G*: Runs machine level program starting at memory (adrs).
- **adrsT** *800T*: Traces a program starting at memory location (adrs) and continues trace until hitting a breakpoint. Break occurs on instruction $00$ (BRK), and returns control to system monitor. Opens 6502 status registers (see note 1).
- **adrsS** *C050S*: Single steps through program beginning at memory location (adrs). Type a letter S for each additional step that you want displayed. Opens 6502 status registers (see Note 1).
- **(Control E)** *EC*: Displays 6502 status registers and opens them for modification (see Note 1).
- **(Control Y)** *YC*: Executes user specified machine language subroutine starting at memory location (3F8).

**Note 1:**
6502 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 = 00  P = 32  S = F2
*: FF  Changes A register only
*:FF 00 33  Changes A, X, and Y registers

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

### Hexadecimal Arithmetic

- **d1+d2** *78+34*: Performs hexadecimal sum of d1 plus d2.
- **d1-d2** *AE-34*: Performs hexadecimal difference of d1 minus d2.
<table>
<thead>
<tr>
<th>Command Format</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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

*1gGl 4gGg Afft  
Multiple monitor commands may be given on same line if separated by a "space".

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

"Control" characters are indicated by a super-scripted "C" such as \( G^C \). They are obtained by holding down the CTRL key while typing the specified letter. Control characters are NOT displayed on the TV screen. \( B^C \) and \( C^C \) must be followed by a carriage return. Screen editing characters are indicated by a sub-scripted "E" such as \( D_e \). They are obtained by pressing and releasing the ESC key then typing specified letter. Edit characters send information only to display screen and does not send data to memory. For example, \( U^C \) moves to cursor to right and copies text while \( A^C \) 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;&lt;-&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&lt;sub&gt;E&lt;/sub&gt;</td>
<td>Move cursor to right</td>
</tr>
<tr>
<td>B&lt;sub&gt;E&lt;/sub&gt;</td>
<td>Move cursor to left</td>
</tr>
<tr>
<td>C&lt;sub&gt;E&lt;/sub&gt;</td>
<td>Move cursor down</td>
</tr>
<tr>
<td>D&lt;sub&gt;E&lt;/sub&gt;</td>
<td>Move cursor up</td>
</tr>
<tr>
<td>E&lt;sub&gt;E&lt;/sub&gt;</td>
<td>Clear text from cursor to end of line</td>
</tr>
<tr>
<td>F&lt;sub&gt;E&lt;/sub&gt;</td>
<td>Clear text from cursor to end of page</td>
</tr>
<tr>
<td>@&lt;sub&gt;E&lt;/sub&gt;</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>C050</td>
<td>10 POKE -16304,0</td>
<td>Set color graphics mode</td>
</tr>
<tr>
<td>C051</td>
<td>20 POKE -16303,0</td>
<td>Set text mode</td>
</tr>
<tr>
<td>C052</td>
<td>30 POKE -16302,0</td>
<td>Clear mixed graphics</td>
</tr>
<tr>
<td>C053</td>
<td>40 POKE -16301,0</td>
<td>Set mixed graphics (4 lines text)</td>
</tr>
<tr>
<td>C054</td>
<td>50 POKE -16300,0</td>
<td>Clear display Page 2 (BASIC commands use Page 1 only)</td>
</tr>
<tr>
<td>C055</td>
<td>60 POKE -16299,0</td>
<td>Set display to Page 2 (alternate)</td>
</tr>
<tr>
<td>C056</td>
<td>70 POKE -16298,0</td>
<td>Clear HIRES graphics mode</td>
</tr>
<tr>
<td>C057</td>
<td>80 POKE -16297,0</td>
<td>Set HIRES graphics mode</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>TEXT Mode Controls</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0020</td>
<td>90 POKE 32,LI</td>
<td>Set left side of scrolling window to location specified by LI 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. LI+W1&lt;40. W1&gt;0</td>
</tr>
<tr>
<td>0022</td>
<td>110 POKE 34,T1</td>
<td>Set window top to line specified by T1 in range of 0 to 23</td>
</tr>
<tr>
<td>0023</td>
<td>120 POKE 35,B1</td>
<td>Set window bottom to line specified by B1 in the range of 0 to 23. B1&gt;T1</td>
</tr>
<tr>
<td>0024</td>
<td>130 CH=PEEK(36)</td>
<td>Read/set cursor horizontal position in the range of 0 to 39. If using TAB, you must add &quot;1&quot; to cursor position read value; Ex. 140 and 150 perform identical function.</td>
</tr>
<tr>
<td></td>
<td>140 POKE 36,CH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>150 TAB(CH+1)</td>
<td></td>
</tr>
<tr>
<td>0025</td>
<td>160 CV=PEEK(37)</td>
<td>Similar to above. Read/set cursor vertical position in the range 0 to 23.</td>
</tr>
<tr>
<td></td>
<td>170 POKE 37,CV</td>
<td></td>
</tr>
<tr>
<td></td>
<td>180 VTAB(CV+1)</td>
<td></td>
</tr>
<tr>
<td>0032</td>
<td>190 POKE 50,127</td>
<td>Set inverse flag if 127 (Ex. 190)</td>
</tr>
<tr>
<td></td>
<td>200 POKE 50,255</td>
<td>Set normal flag if 255(Ex. 200)</td>
</tr>
<tr>
<td>FC58</td>
<td>210 CALL -936</td>
<td>(@F) Home cursor, clear screen</td>
</tr>
<tr>
<td>FC42</td>
<td>220 CALL -958</td>
<td>(F) 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>(E&lt;sub&gt;E&lt;/sub&gt;) Clear from cursor to end of line</td>
</tr>
<tr>
<td>FC66</td>
<td>240 CALL -922</td>
<td>(J&lt;sup&gt;C&lt;/sup&gt;) 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>Step</th>
<th>BASIC Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C030</td>
<td>360 X=PEEK(-16336)</td>
<td>Toggle speaker</td>
</tr>
<tr>
<td></td>
<td>365 POKE -16336,0</td>
<td></td>
</tr>
<tr>
<td>C000</td>
<td>370 X=PEEK(-16384)</td>
<td>Read keyboard; if X&gt;127 then key was pressed.</td>
</tr>
<tr>
<td>C010</td>
<td>380 POKE -16368,0</td>
<td>Clear keyboard strobe – always after reading keyboard.</td>
</tr>
<tr>
<td>C061</td>
<td>390 X=PEEK(16287)</td>
<td>Read PDL(0) push button switch. If X&gt;127 then switch is &quot;on&quot;.</td>
</tr>
<tr>
<td>C062</td>
<td>400 X=PEEK(-16286)</td>
<td>Read PDL(1) push button switch.</td>
</tr>
<tr>
<td>C063</td>
<td>410 X=PEEK(-16285)</td>
<td>Read PDL(2) push button switch.</td>
</tr>
<tr>
<td>C058</td>
<td>420 POKE -16296,0</td>
<td>Clear Game I/O AN&lt;sub&gt;0&lt;/sub&gt; output</td>
</tr>
<tr>
<td>C059</td>
<td>430 POKE -16295,0</td>
<td>Set Game I/O AN&lt;sub&gt;0&lt;/sub&gt; output</td>
</tr>
<tr>
<td>C05A</td>
<td>440 POKE -16294,0</td>
<td>Clear Game I/O AN&lt;sub&gt;1&lt;/sub&gt; output</td>
</tr>
<tr>
<td>C05B</td>
<td>450 POKE -16293,0</td>
<td>Set Game I/O AN&lt;sub&gt;1&lt;/sub&gt; output</td>
</tr>
<tr>
<td>C05C</td>
<td>460 POKE -16292,0</td>
<td>Clear Game I/O AN&lt;sub&gt;2&lt;/sub&gt; output</td>
</tr>
<tr>
<td>C05D</td>
<td>470 POKE -16291,0</td>
<td>Set Game I/O AN&lt;sub&gt;2&lt;/sub&gt; output</td>
</tr>
<tr>
<td>C05E</td>
<td>480 POKE -16290,0</td>
<td>Clear Game I/O AN&lt;sub&gt;3&lt;/sub&gt; output</td>
</tr>
<tr>
<td>C05F</td>
<td>490 POKE -16289,0</td>
<td>Set Game I/O AN&lt;sub&gt;3&lt;/sub&gt; output</td>
</tr>
<tr>
<td>NAME</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>TITLE</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>loc0</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>loc1</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>wndlft</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>wndvth</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>wndtop</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>wndtm</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>ch</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>cv</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>grasl</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>grash</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>basl</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>bash</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>bas2l</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>bas2h</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>m2</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>lnnem</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>rtln</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>v2</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>rnmem</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>rtnh</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>mask</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>chksum</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>format</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>lastin</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>length</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>sign</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>color</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>invflg</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>prompt</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>ysav</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>ysav1</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>cswl</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>cswh</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>kswl</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>kswh</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>pcl</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>pch</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>xqt</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>ail</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>ailh</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>a2l</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>a2h</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>a3l</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>a3h</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>a4l</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>a4h</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>a5l</td>
<td>EPZ</td>
<td></td>
</tr>
<tr>
<td>a5h</td>
<td>EPZ</td>
<td></td>
</tr>
</tbody>
</table>
F858: A0  ASL A
F859: A0  ASL A
F85A: 05 26 ORA GRASL
F85C: 85 26 STA GRASL
F85E: 60 RTS
F85F: A5 30 XX1COL LDA COLOR INCREMENT COLOR BY 3
F861: 13 CLC
F862: 69 03 ADC #101
F864: 29 OF SETCOL AND #$0F SETS COLOR = #17 # MOD 16
F866: 66 30 STA COLOR
F868: 0A ASL A
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 SCN LSR A READ SCREEN Y-COORD/2
F872: 06 PHP SAVE LSB (CARRY)
F873: 20 47 F8 JSR GRASCALC CALC BASE ADDRESS
F876: 81 26 LDA (GS)+L, Y GET BYTE
F878: 28 PLP RESTORE LSB FROM CARRY
F879: 90 04 SCN2 RLC #07 IF EVEN, USE LO H
F87D: 4A LSR A
F87C: 4A LSR A
F87D: 4A LSR A
F87E: 4A LSR A
F87F: 29 0F HIONLY AND #$0F MASK 4-BITS
F881: 60 RTS
F882: A6 3A INDSOI LDY PCL PRINT PCL,1
F884: A4 3E LDY PCV
F886: 20 96 FD JSP PXX2
F889: 20 48 F9 JSR PRINTX FOLLOWED BY A BLANK
F88C: A1 3A LCA (PCL,X) GET OF CODE
F88E: A8 INDS2 LSRY SHIFT HIGH-HALF BYTE DOWN
F891: 4A LSR A
F894: 90 09 RLC UVEC EVEN BIT 1 TEST
F895: 67 ROP A BCS THP XXXX11 INVALID OF
F896: C9 12 CMP #$A2 EVEN RETN ERR OPCODE 5% INVALID
F897: F0 UC ROR ERR OPCODE 5% INVALID
F898: 29 67 AND #$17 TASK FST
F89A: 4A TEC EVEN LSR INTO CARRY FOR L/R TEST
F89C: AA TAX
F89D: E0 62 F5 LDA PLYT1,X GET FORMAT INDEX BYTE
F8A0: 20 75 F6 JSR SCN2 H-RYPE ON CARRY
F8A3: D0 04 BNE GETTAB
F8A5: A0 80 ERP LDY #80 SUBSTITUTE SKB FOR INVALID OPS
F8A7: A9 00 EYP LDA #$00 SFT PRINT FORMAT INDEX TO 0
F8A9: AA GETTAB TAX
F8AA: BD A9 F9 LDA PLYT2,X INDEX INTO PRINT FORMAT TABLE
F8AB: 85 22 STA FORMAT SAVE FOR ADP FIELD FORMATTING
F8AF: 29 03 #03 AND #$03 MASK FOR 2-BIT LENGTH
F8BD: 85 2F STA LENGTH
F8BE: 98 TTA OPCODE
F8BF: 29 8F AND #$8F MASK FOR 1XXX1010 TEST
F8C0: 98 TAX SAVF IT
F8C2: AA TYA OPCODE TO A AGAIN
F8C8: 4A MNNDX3 LSR A FORM INDEX INTO MNEMONIC TABLE
F8CC: 4A LSR A
F8CD: 24 A MNNDX2 LSR A
F8CE: 88 DEY 1) XXXX1010=>00101XXX
F8C8: 20 22 ORA #$20 2) XXXX1111=>00111XXX
F8C9: 88 DEY 3) XXXX1000=>00100XXX
F8CA: D0 F2 BNE MNNDX3 4) XXXX1000=>00100XXX
F8CC: 60 RTS
F8CD: FF FF FF DFB $FF,$FF,$FF
F8D0: 20 82 F8 INSTDSP JSR INSOSI GEN FMT, LEN BYTES
F8D3: 48 PLA SAVE MNEMONIC TABLE INDEX
F8D4: B1 3A PRNTOP LDA (PCL),Y
F8D6: 20 DA FD JSR PRYTYE PRINT 2 BLANKS
F8DB: 3A 0F LDA #$0F SAVE DUMMY VALUE
F8E0: C8 CPY LENGTH PRINT INST (1-3 BYTES)
F8E0: C8 CPY LENGTH PRINT INST (1-3 BYTES)
F8E1: 90 F1 BCC PRNTPrint CHAR COUNT FOR MNEMONIC PRINT
LDA A9 00
LDA B5 41 STA A3H
LDA A2 FB LDX #5FB
LDA A9 A0 RDSP1 LDA #$A0
LDA B8 FA STA A3H
LDA B6 PD JSR COUT
LDA B9 E FA STA RTBL-$FB,X
LDA B6 PD JSR COUT
LDA A9 BD LDA #$BD
STA BD PD JSR COUT
LDA B5 4A LDA ACC+5,X
JSR PROYTE
LDA B8 INX
LDA 3B EA BMI RDSP1
RTS
LDA 18 BRANCH CLC BRANCH TAKEN,
LDA A0 01 LDX #$01 ADD LEN+2 TO PC
LDA B1 3A LDA (PCL),Y
LDA 20 56 F9 JSR PCADJ3
LDA B5 3A STA PCL
LDA 98 TYA
LDA 38 SEC
LDA B0 A2 BCS PCINC2
LDA 20 4A FF NRPNCH JSR SAVE NORMAL RETURN AFTER
LDA 38 SEC XEO USER OF
LDA B0 9E PCS PCINC3 GO UPDATE PC
LDA EA 9F INITBL NOP
LDA EA 9F NOP DUMMY FILL FOR
LDA 1E 4C 0B FB JMP NRPNCH XEO AREA
LDA 4C FD FA JMP Branch
LDA B1 C1 RTBL DPP SCI
LDA B1 D8 DBP $D8
LDA B1 D9 DBP $D9
LDA B1 D0 DBP $D0
LDA B1 D3 DBP $D3
LDA B1 A0 70 C0 PREAD LDA PTPFG TRIGGER PADDLES
LDA B2 00 LDX #$00 INIT COUNT
LDA EA 9F NOP COMPENSATE FOR 1ST COUNT
LDA EA 9F NOP
LDA B5 64 C0 PREAD2 LDA PADDLO,X COUNT Y-PEG EVERY
LDA 10 04 BPL RTS2D 12 USSEC
LDA C8 INY
LDA D0 F8 BNE PREAD2 EXIT AT 255 MAX
LDA 88 DEY
LDA 60 RTS2D RTS
LDA A9 00 INIT LDA #$00 CLR STATUS FOR DEBUG
LDA B5 4B STA STATUS SOFTWARE
LDA B3 56 C0 LDA LORIES
LDA B3 54 C0 LDA LOKSCR INIT VIDEO MODE
LDA B3 51 C0 SETTXT LDA TXTSET SET FOR TEXT MODE
LDA B3 C0 LDA #$00 FULL SCREEN WINDOW
LDA B3 F0 0B REO SETWND
LDA B4 50 C0 SETGR LDA TXTCLR SET FOR GRAPHICS MODE
LDA B4 53 C0 LDA MIXSET LOWER 4 LINES AS
LDA 20 36 F8 JSR CLPTOP TEXT WINDOW
LDA B4 1A 14 LDA #$14
LDA B4 22 SETWND STA WNDTOP SET FOR 40 COL WINDOW
LDA A9 00 LDA #$00 TOP IN A-PGC,
LDA B4 85 20 STA WNDLFT BTM AT LINE 24
LDA B5 31 21 LDA #$16
LDA B5 18 LDA #$00
LDA B5 23 STA WNDRTM VTAP TO ROW 23
LDA B5 19 LDA #$17
LDA B5 25 TABV STA CV VTAB TO ROW IN A-REG
LDA B5 D1 4C 22 FC JMP VTAB
LDA 20 A5 FB MUL* JSP #$01
LDA B6 40 A0 10 MUL JSP #$01 ABS VAL OF AC AUX
LDA B6 41 A0 10 MUL LDY #$10 INDEX FOR 16 BITS
LDA B6 5A 50 MUL2 LDA #ACL ACK + AUX + XTND
LDA B6 7A 4A LSR A TO AC, XTND
LDA B6 90 0C BCC MUL4 IF NO CARRY,
LDA B6 A5 18 CLC MUL4 NO PARTIAL PROD.
LDA B6 A2 FE LDX #$FE
LDA B6 5A 54 MUL3 LDA XTNDL+2,X ADD MPLCND (AUX)
LDA B6 7A 56 AOC AUXL+2,X TO PARTIAL PROD
LDA B7 95 54 STA XTNDL+2,X (XTND).
LDA B7 3E 18 INX
LDA B7 D1 D0 F7 BNV "MUL 3
LDA B7 6A 20 03 MUL4 LDX #$03
LDA B7 76 MUL5 DPP #$76
LDA B7 90 50 CPF #$50
LDA B7 A1 CA DEX
LDA B7 8A 10 FP 3PL MUL5
LDA B7 D1 88 BCP DEX
LDA B7 D1 E5 BNE MUL2
LDA B8 60 BCT
FF66: 20 75 FE GO  JSR A1PC  AND PC IF SPEC'D
FF69: 20 3F FF  JSR RESTORE  RESTORE META PEGS
FEBC: 6C 3A 00  JSR (PCL)  GO TO USER SURR
FEBF: 4C 07 FA REGZ  JMP REGSP  TC REG DISPLAY
FEC2: C6 34  TPACF  DEC YSAY
FEC4: 20 75 FE STEPS  JSR A1PC  AND TO PC IF SPEC'D
FEC7: 4C 43 FA JSR STEP  TAKE ONE STEP
FECA: 4C F6 03 USP  JSR USPADDR  TO USP SURR AT USPADDR
FECD: A9 40 WRITE  LDA #$40
FEE: 20 CS FC JSR HEADER  WRITE 10-SFC HEADER
FEDE: A0 27 LDY #$27
FEED: A2 00 #R1  LDX #$00
FEE6: 41 3C EOP (ALL, X)
FED8: 48 PLA
FED9: A1 3C LDA (ALL, X)
FEDB: 20 BA FC JSR NXT1
FEED: A0 1D LDY #$10
FEE3: 68 PLA
FEE4: 90 2E EEC  R1
FEE6: A0 22 LDY #$22
FEE8: 20 ED FE JSR WRBYTE
FEEB: F0 40 SEC PELL
FEED: A2 10 WRBYTE  LDX #$10
FEEF: 0A 1B WRBYTE2  ASL A
FEOF: 20 D6 FC JSR WRBIT
FEF3: D0 FA BNE WRBYTE2
FEF5: 60 RTS
FEF6: 20 00 FE CRMON  JSR BL1  HANDLE CR AS PLANK
FEF9: 68 PLA  AND P1N TO MON
FEFA: 68 PLA  AND P1N TO MON
FEB8: D0 6C BNE W1N2
FEBD: 20 FA FC READ  JSR ROZBIT  FIND TAPEIN EDGE
FPOD: A9 16 LDA #$16
FP02: 20 C9 FC JSR HEADER  DMAY 3.5 SECONDS
FP05: 85 2E STA CHKSUM INIT CHKSUM=0FF
FP07: 20 FA FC JSR ROZBIT  FIND TAPEIN EDGE
FP0A: A0 24 RD2  LDY #$24  LOOK FOR SYNC VIT
FP0C: 20 ED FC JSR ROZBIT  (SHORT 0)
FP0F: B0 F9 BCS RD2  LOOP UNTIL FOUND
FP11: 20 FD FC JSR ROZBIT  SKIP SECOND SYNC H-CYCLE
FP14: A0 3B LDY #$3B  INDEX FOP 0/1 TEST
FP16: 20 FC FC RD3  JSR ROZBIT  READ A BYTE
FP19: 81 3C STA (A1L, X) STORE AT (A1)
FP1B: 45 2E EOR CHKSUM
FP1D: 85 2E STA CHKSUM  UPDATE RUNING CHKSUM
FP1F: 20 3A FC JSR NXT1  INCH A1, COMPARE TO A2
FP22: A0 35 LDY #$35  COMPENSATE 0/1 INDEX
FP24: 90 F0 PCC RD3  LOOP UNTIL DONE
FP26: 20 EC FC JSR ROBYTE  READ CHKSUM BYTE
FP29: C5 2E CMP CHKSUM
FP2B: F0 00 SEP SKILL  GOOD, SOUND BELL AND RETURN
FP2D: A9 C5 PPERR  LDA #$C5
FP2F: 20 ED FD JSR COUT  PRINT "ERR", THEN BELL
FP32: A9 D2 LDA #$C2
FP34: 20 ED FD JSR COUT
FP37: 20 ED FD JSR COUT
FP3A: A9 87 BELL  LDA #$67  OUTPUT BELL AND RETURN
FP3C: 4C ED FD JSR COUT
FP3F: A5 48 RESTORE  LDA STATUS  RESTORE 6502 PEG CONTENTS
FP41: 48 PEA  USED BY DEBUG SOFTWARE
FP44: A5 45 LDA ACC
FP46: A6 46 RESTRI  LDX XREG
FP48: A4 47 LDY YPEG
FP4F: 28 PLP
FP4F: 60 RTS
FP4A: 85 45 SAVE  STA ACC  SAVE 6502 REG CONTENTS
FP4C: 86 46 SAV1  STX XPEG
FP4E: 64 47 STY YPEG
FP50: 08 PHP
FP51: 68 PLA
FP52: 85 48 STA STATUS
FP54: 8A TSX
FP55: 86 45 STX SPNT
FP57: DO COLD
FP59: F0 60 PFS
FP59: 20 84 FE PESET  JSR SETNORM  SET SCREEN MODE
FP5C: 20 2F FF JSR INIT  AND INIT KBD/SCREEN
FP5F: 20 93 FE JSR SETVID  AS I/O DEV'S
FP62: 20 89 FE JSR SETKIND
FP65: D8 MON  CLD
FP66: 20 3A FF JSR BELL
FP69: A9 AA MONZ  LDA #$3A  "" PROPT FOR MON
FP6B: 85 33 STA PROMPT
FP6D: 20 47 FD JSR DEILNMZ  READ A LINE

88
<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFF0: 83</td>
<td>DFB</td>
<td>SETNORM-1</td>
</tr>
<tr>
<td>FFF1: 7F</td>
<td>DFB</td>
<td>SETINV-1</td>
</tr>
<tr>
<td>FFF2: 50</td>
<td>DFB</td>
<td>LIST-1</td>
</tr>
<tr>
<td>FFF3: CC</td>
<td>DFB</td>
<td>WRITE-1</td>
</tr>
<tr>
<td>FFF4: F5</td>
<td>DFB</td>
<td>GO-1</td>
</tr>
<tr>
<td>FFF5: FC</td>
<td>DFB</td>
<td>READ-1</td>
</tr>
<tr>
<td>FFF6: 17</td>
<td>DFB</td>
<td>SETMODE-1</td>
</tr>
<tr>
<td>FFF7: 17</td>
<td>DFB</td>
<td>SFTMODE-1</td>
</tr>
<tr>
<td>FFF8: F5</td>
<td>DFB</td>
<td>CRMON-1</td>
</tr>
<tr>
<td>FFF9: 03</td>
<td>DFB</td>
<td>BLANK-1</td>
</tr>
<tr>
<td>FFFA: FA</td>
<td>DFB</td>
<td>NMI</td>
</tr>
<tr>
<td>FFFB: 03</td>
<td>DFB</td>
<td>NMI/256</td>
</tr>
<tr>
<td>FFFC: 59</td>
<td>DFB</td>
<td>RESET</td>
</tr>
<tr>
<td>FFFD: FF</td>
<td>DFB</td>
<td>RESET/256</td>
</tr>
<tr>
<td>FFFE: B6</td>
<td>DFB</td>
<td>IRC</td>
</tr>
<tr>
<td>FFFF: FA</td>
<td>XOUTC</td>
<td>EQU</td>
</tr>
</tbody>
</table>
***************
APPLE-II
MINI-ASSEMBLER
***************

TITLE "APPLE-II MINI-ASSEMBLER"
FORMATT EPZ $2F
LENGTH EPZ $2F
MODE EPZ $31
FORMAT EPZ $13
YSAV EPZ $34
L EPZ $35
PCL EPZ $3A
PCOM EPZ $3B
ALW EPZ $3D
A2L TPZ $3E
A2L EPZ $3F
A4L EPZ $42
A4R EPZ $43
PMT EPZ $44
I1 EPZ $200
INSG2 ECU $F86E
INSGSP ECU $F860
PRLJL2 ECU $F94A
PCAMJ ECU $F953
CHAR1 ECU $F944
CHAR2 ECU $F94A
MENDL ECU $F9C0
WALKAP ECU $F000
CURSUP ZOU $F01A
GETLENZ ECU $F067
CONST ECU $F0E0
M1 ECU $F0E0
ALCPLP ECU $F07A
ELL ECU $FF3A
GEMU ECU $FFA7
TOSUP ECU $FFEF
CODE ECU $FFCC

F500: EN 1 REL SF C P501  IS NOT COMPATIBLE
F502: 4A REL ISP A WITH RELATIVE CODE
F503: 5O 14 DLY FPR3
F505: A4 3F DLY A2L
F507: A6 3F LDX A2L DOUBLE OPCRT 3ST
F509: D0 01 BXE REL2
F50B: 88 DEX
F50C: CA REL2 TXA
F50D: 8A TXA
F50E: 18 CLC
F50F: E5 3A SRC PCL FORM ADDR-PC-2
F511: 85 3E STA A2L
F513: 10 01 BPL REL3
F515: C8 INY
F516: 98 REL3 TYA

91
F517: E5 3B SBC PCH
F518: D0 6B ERR3 BNE ERR ERROR IF >1-BYTE BRANCH
F519: A4 2F FINDOP LDY Y INDEX
F51A: B9 83 0D FINDOP2 LDA A1,Y MOVE INST TO (PC)
F51B: 81 3A STA (PCL) Y
F522: 88 DEY
F523: 10 F8 BPL FNDOP2
F524: 20 1A FC JSR CURSUP JSR CURSUP
F525: 20 1A PC JSP CURSUP RESTORE CURSOR
F526: 20 00 08 JSR INSTDSP TYPE FORMATTED LINE
F527: 20 53 F9 JSR PCDATA UPDATE PC
F531: 84 3B STY PCH
F532: 85 3A SPA PCL
F535: 4C 95 F5 JMP NEXTLINE GET NEXT LINE
F538: 20 BE FF FAKEMON3 JSR TOSUB GO TO DELIM HANDLER
F53D: 20 A7 FF FAKEMON JSR GETNUM READ PARAM
F53E: 20 34 3A STY YSAV SAVE Y-INDEX
F542: A0 17 LDY #$17 INIT DELIMITER INDEX
F544: 88 FAKEMON2 DEY CHECK NEXT DELIM
F545: 30 4B BMI FSETT ERR IF UNRECOGNIZED DELIM
F547: D9 CC FF CMP Chris TEXT COMPARE WITH DELIM TABLE
F54A: D0 F8 BNE FAKEMON2 NO MATCH
F54C: C0 15 CPY #$15 MATCH, IS IT CR?
F54E: D0 E8 BNE FAKEMON3 NO, HANDLE IT IN MONITOR
F550: A5 31 LDA MODE
F552: A0 00 LDY #$0
F554: C6 34 DEC YSAV
F556: 20 00 FE JSR BL1 HANDLE CR OUTSIDE MONITOR
F559: 4C 95 F5 JMP NEXTLINE
F55C: A5 30 3B TRYNEXT JMP A1H 3D GET TRIAL OPCODE
F55E: 20 8E F8 JSR IMSDS2 GET FMT+LENGTH FOR OPCODE
F561: AA TAX
F562: BD 00 FA LDA MNEMR,X GET LOWER MNEMONIC BYTE
F565: C5 42 CMP A4L MATCH?
F567: D0 01 13 BNE NEXTOP NO, TRY NEXT OPCODE
F569: BD C0 F9 LDA MNEML,X GET UPPER MNEMONIC BYTE
F56C: C5 43 CMP A4H MATCH?
F56E: D0 0C LDA MNEMR,X NO, TRY NEXT OPCODE.
F570: A5 44 44 LDA PNT
F572: A4 2E LDY FORMAT GET TRIAL FORMAT
F574: C0 9D CPY #$9D TRIAL FORMAT RELATIVE?
F576: F0 88 BEQ REL YES.
F578: C5 2E NREL CMP FORMAT SAME FORMAT?
F57A: F0 9F BEQ FINDOP YES.
F57C: C6 3D NEXTOP DEC A1H NO, TRY NEXT OPCODE
F57E: D0 DC BNE TRYNEXT
F580: E6 44 INC PNT NO MORE, TRY WITH LEN=2
F582: C6 35 DEC L WAS L=2 ALREADY?
F584: F0 6D BEQ TRYNEXT NO.
F586: A4 34 ERR LDY YSAV YES, UNRECOGNIZED INST.
F588: 98 ERR2 TAY
F589: AA TAX
F58A: 20 4A F9 JSR PSBL2 PRINT " UNDER LAST READ
F58D: A9 DE LDA #$DE CHAR TO INDICATE ERROR
F58F: 20 ED FD JSR COUT POSITION.
F592: 20 3A FF RESSET2 JSR BELL
F595: A9 A1 NXLIME LDA #$A1 "!
F597: B5 33 STA PROMPT INITIALIZATION PROMPT
F599: 20 67 FD JSR GETIN2 GET LINE.
F59C: 20 C7 FF JSR 2MODE INITIAL SCREEN STUFF
F59F: AD 00 02 LDA IN GET CHAR
F5A2: C9 A0 CMP #$A0 ASCII BLANK?
F5A4: F0 13 BEO SPACE YES
F5A6: C8 INY
F5A7: C9 A4 CMP #$A4 ASCII "$" IN COL 1?
F5A9: F0 92 BEQ FAKEMON YES, SIMULATE MONITOR
F5AB: 88 DEY NO, BACKUP A CHAR
F5AC: 20 A7 FF JSR GETNUM GET A NUMBER
F5AF: C9 93 CMP #$93 "!" TERMINATOR?
F5B1: D0 D5 ERR4 BNE ERR2 NO, ERR.
F5B3: 8A TXA
F5B4: F0 D2 BEO ERR2 NO ADR PRECEDING COLON.
F5B6: 20 7F FE JSR ALPCLP MOVE ADR TO PCL, PCH.
F5B9: A9 03 SPACE LDA #$3 COUNT OF CHAR IN MNEMONIC
F5BB: 85 3D STA A1H
F5BD: 20 34 3B NXMN JSR GETNSP GET FIRST MNEM CHAR.
F5C0: 0A NXM ASL A
F5C1: E9 BE SEC #$RE SUBTRACT OFFSET
F5C3: C9 C2 CMP #$SC2 LEGAL CHAR?
F5C5: 90 C1 BCC ERR2 NO.
F5C7: 0A ASL A COPPRESS-LEFT JUSTIFY
F5CB: 0A ASL A
F5CC: 0A NXM2 ASL A DO 5 TRIPLE WORD SHIFTS
*************
* APPLE-II FLOATING *
* POINT ROUTINES *
* COPYRIGHT 1977 BY *
* APPLE COMPUTER INC. *
* ALL RIGHTS RESERVED *
* S. WOZNIAK *
*************

TITLE "FLOATING POINT ROUTINES"

SIGN EPI SF3
X2 EPI SF4
M2 EPI SF5
X1 EPI SF6
M1 EPI SF7
E EPI SFC

GVLOR EQU SF35

F425: 18 ADD CLC CLEAR CARRY.
F426: A2 02 LDX #2 INDEX FOR 3-BYTE ADD.
F428: B5 9F ADOI LDA #1,X AND A BYTE OF MANT2 TO MANT1.
F42A: 75 F5 ADC #2,X ADD A VAL OF M1, THEN SWAP WITH M2
F42C: 95 F9 STA #1,X INDEX TO NEXT MORE SIGNIF. BYTE.
F42E: CA DEFY LOOP UNTIL DONE.
F42F: 10 F7 DPL ADD1
F431: 60 RTS RETURN.
F432: 06 F3 MD1 ASL SIGN CLEAR LSE OF SIGN.
F434: 20 37 F4 JSR ABS/3P ABS VAL OF M1, THEN SWAP WITH M2
F437: 24 F9 ABS/A1 BIT #1 MANT1 NEGATIVE?
F439: 10 05 BPL ABS/A1 NO, SWAP WITH MANT2 AND RETURN.
F43B: 20 A4 F4 JSK DEC=PL YES, COMPLEMENT IT.
F43E: 66 F3 INC SIGN INC SIGN, COMPLEMENTING LSP.
F440: 3F ABS/A1 SEC SET CARRY FOR RETURN TO MUL/DIV.
F441: A2 04 SKP LDX #4 INDEX FOR 4-BYTE SWAP.
F443: 94 FB SWP1 STY #1,X
F445: B5 F7 LDA X1-1,X SWAP A BYTE OF EXP/MANT1 WITH
F447: 94 F3 LDY X2-1,X EXP/MANT2 AND LEAVE A COPY OF
F449: 94 F7 STY X1-1,X MANT1 IN L (3 BYTES). F43 USED
F44B: 95 F3 STA Y2-1,X
F44D: CA DEFY ADVANCE INDEX TO NEXT BYTE.
F44E: DD F3 SWP1 LOOP UNTIL DONE.
F450: 60 RTS RETURN:
F451: 45 E5 FLOAT LDA #5F INIT EXP1 TO 14,
F453: 83 F6 STA Y1 THEN NORMALIZE TO FLOAT.
F455: A5 F9 '0'/M1 LDA M1 HIGH-ORDER MANT1 BYTE.
F457: C9 C0 CMP #5C0 UPPER TO FITS UNEXPECTED?
F459: 30 0C RLI INC1 YER, RETURN WITH MANT1 NORMALIZED
F45B: C6 46 DEC X1 DISPENS EXP1.
F45C: 06 FF ASL M1+2
F45F: 26 FA RCL M1+1 SHIFT MANT1 (3 BYTES) LEFT.
F461: 26 F9 RCL M1
F463: A5 F8 NORM LDA XI EXP1 ZERO?
F465: D0 EE BNE NORM=1 NO, CONTINUE NORMALIZING.
F467: 60 RTS1 PTS RETURN.
F468: 20 A4 F4 FSUB JSR FCMP=A1 CMPL MANT1, CLEAR CARRY UNLESS 0
F46B: 20 7B FA SWAPEG JSR ALG/WiG RIGHT SHIFT MANT1 OR SWAP WITH
F46E: A5 F4 ADDA LDA X2
F470: C5 F8 CMP X' COMPARE EXP1 WITH EXP2.
F472: D0 F7 BNE SWAPEL IF #, SWAP ADDENDS OR ALIGN MANTS.
F474: 20 25 F4 JSR ADD ADD ALIGNED MANTISSAS.
F477: 50 EA ADDEND BSE NORM NO OVERFLOW, NORMALIZE RESULT.
F479: 70 05 BVS BYLOG OV: SHIFT XI RIGHT, CARRY INTO SIGN
F689:  20  4A  FF  SAVE  S16  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  JSP  SW16C  INTERRUPT  AND  EXECUTE
F695:  4C  92  F6  JSR  SW16E  ONE  SWEET16  INTR.
F698:  E1  1E  SW16C  INC  R15L  INCP  SWEET16  PC  FOR  FETCH
F69A:  D0  02  BHE  SW16D  INCP  SWEET16  PC  FOR  FETCH
F69C:  E6  1F  INC  R15H
F69E:  A9  F7  SW16D  LDA  #S16PAG
F6A0:  49  FFA  SAVE  R15L
F6A1:  A0  00  LDY  #50
F6A3:  B1  1E  LDA  (R15L),Y  FETCH  INSTR
F6A5:  29  0F  AND  #SF  MASK  REG  SPECIFICATION
F6A7:  0A  ARL  A  DOUBLE  FOR  2-BYTE  REGISTERS
F6AB:  AA  TAX  TO  X-REG  FOR  INDEXING
F6A9:  4A  LSR  A
F6AA:  51  1E  EOR  (P15L),Y  NON  HAVE  OPCODE
F6AC:  F0  0B  BEQ  TOSR  IF  ZERO  THEN  NON-REG  OP
F6AD:  86  1D  STX  R14H  INDICATE'PRIOR  RESULT  REG'
F6B0:  4A  LSR  A
F6B1:  4A  LSR  A  OPCODE*2  TO  LSP'S
F6B2:  4A  LSR  A
F6B3:  A8  DAA  TO  Y-REG  FOR  INDEXING
F6B4:  B9  F1  F6  LDA  CPTBL-2,Y  LOW-ORDER  ADR  BYTE
F6B7:  48  PHA  INTO  STACK
F6B8:  60  RTS  INTO  REG-OP  ROUTINE
F6B9:  E6  1E  TO3P  INC  R15L  INC  PC
F6BB:  DD  02  BME  R0=m+2  INC  PC
F6BD:  E6  1F  INC  #15H
F6BF:  40  F4  F6  TLACS  LDA  PPMTL,X  LOW-ORDER  ADR  BYTE
F6C2:  46  PHA  INTO  STACK  FOR  NON-REG  OP
F6C3:  A5  1D  LDA  R14H  'PRIOR  RESULT  REG'  INDEX
F6C5:  4A  LSR  A  PREPARE  CARRY  FOR  PC,  INC.
F6C6:  60  BIS  CCTO  NON-REG  OP  ROUTINE
F6C7:  68  RTNZ  PLA  TOP  RETURN  ADDRESS
F6C8:  68  PLA
F6C9:  20  3F  FF  JSR  RESTORE  RESTORE  6502  REG  CONTENTS
F6CC:  6C  1E  00  JMP  (P15L)  RETURN  TO  6502  CODE  VIA  PC
F6CF:  E1  1F  SETZ  LDA  (P15L),Y  HIGH-ORDER  BYTE  OF  CONSTANT
F601: 95 01 STA R0H,X
F603: 88 DEY
F604: B1 1E LDA (R15L),Y LOW-ORDER BYTE OF CONSTANT
F606: 95 00 STA R0L,X
F608: 98 TYA Y-REG CONTAINS 1
F609: 38 INC
F60A: 65 1E ADC R15L ADD 2 TO PC
F60C: 85 1E STA R15L
F60E: 90 02 BCC SET2
F610: E6 1F INC #15H
F612: 60 RTS
F613: 02 OPTRL
F614: F9 SRPL DFE SET1-1 (1X)
F61A: F9 DFE PM1-1 (0)
F61C: 04 DFE LD1-1 (2X)
F61F: 90 DFE OR1-1 (1)
F624: 0D DFP ST1-1 (3X)
F626: 9E DFP MN1-1 (2)
F628: 25 DFP LDAT1-1 (4X)
F629: AF DFP SC1-1 (3)
F62B: 16 DFP STAT1-1 (5X)
F62C: B2 DFP BP1-1 (4)
F62D: 47 DFP LDDAT1-1 (6X)
F62E: B9 DFP MN1-1 (5)
F62F: 51 DFP STD1-1 (7X)
F630: C0 DFP RS1-1 (6)
F631: 2F DFP POP1-1 (8X)
F632: C9 DFP SN1-1 (7)
F633: 5E DFP STAT1-1 (9X)
F635: 85 DFP BM1-1 (8)
F636: DD DFP ADD1-1 (AX)
F637: 6E DFP BMN1-1-1 (9)
F638: D0 DFP SUS1-1 (BX)
F639: 05 DFP RX1-1 (A)
F63A: 33 DFP MBD1-1 (CX)
F63B: E8 DFP RS1-1 (E)
F63C: 70 DFP CPCR1-1 (DX)
F63D: 93 DFP SS1-1 (C)
F63F: 1E DFP INR1-1 (FX)
F64E: E7 DFP NUL1-1 (D)
F64F: 65 DFP DCR1-1 (FX)
F700: E7 DFP NUL1-1 (E)
F701: E7 DFP NUL1-1 (UNUSED)
F702: E7 DFP NUL1-1 (F)
F703: 10 CA SET BPL SET2 ALWAYS TAKEN
F705: B5 00 LD LDA ROL,X
F707: 85 00 STA ROL
F709: B5 01 STA R0H,X MOVE RX TO R0
F70B: 85 01 STA R0H
F70D: 60 RTS
F70E: A5 00 ST LDA ROL
F710: 95 00 STA R0L,X MOVE R0 TO RX
F712: A5 01 LDA R0H
F714: 95 01 STA R0H,X
F716: 60 RTS
F717: A5 00 STAT LDA ROL
F719: 81 00 STAT2 STA (R0L,X) STORE BYTE INDIRECT
F71B: A0 00 LDY #50
F71D: 84 1D STAT3 STY #14H INDICATE R0 IS RESULT REG
F71F: F6 00 INR INC ROL,X
F721: D0 02 NNE INR2 IMCR RX
F723: F6 01 INC R0H,X
F725: 60 RTS
F726: A1 00 LDAT LDA (R0L,X) LOAD INDIRECT (RX)
F728: B5 00 STA ROL 10 R0
F72A: A0 00 LDY #50
F72C: 84 01 STY R0H,01 ZERO HIGH-ORDER PO BYTE
F72E: F0 ED BEO STAT3 ALWAYS TAKEN
F730: A0 00 POP LDY #50 HIGH ORDER BYTE = 0
F732: F0 06 BEO POP2 ALWAYS TAKEN
F734: 20 66 F7 POPD JSR DCR DECX RX
F737: A1 00 LDA (R0L,X) POP HIGH-ORDER BYTE #RX
F739: A8 TAY SAVE IN Y-REG
F73A: 20 66 F7 POP2 JSR DCP DECX RX
F73D: A1 00 LDA (R0L,X) LOW-ORDER BYTE
F73F: B5 00 STA R0L TO R0
F741: 84 01 STA R0H
F743: A0 00 POP3 LDY #50 INDICATE R0 AS LAST RSLT REG
F745: 84 1D STY #14H
F747: 60 RTS
F748: 20 26 F7 LDDAT JSR LDAT LOW-ORDER BYTE TO R0, INCR RX
F74A: A1 00 LDA (R0L,X) LOW-ORDER BYTE TO R0
F74D: 85 01 STA R0H
F74F: 4C 1F F7 JMP INR IMCR RX
F752: 20 17 F7 STDAT JSR STAT STORE INDIRECT LOW-ORDER
F755: A5 01 LDA ROH BYTE AND INCX RX. THEN
F757: B3 00 STA (R0L,X) STORE HIGH-ORDER BYTE.
F759: 4C 1F F7 JMP INX INCX #Y AND RETURN
F75C: 20 66 F7 STPAT JSP DCY DECP RF
F75F: A5 00 LDA ROL
F761: 83 00 STA (R0L,X) STORE RO LOW BYTE #RX
F763: 4C 43 F7 JMP POP3 INDICATE RO AS LAST LAST REG
F766: B5 00 DCP LDA ROL,X
F768: D0 02 BNE DCR2 DECP RF
F76A: D6 01 DEC ROL,X
F76C: D6 00 DCH2 DEC ROL,X
F76E: 60 RTE
F76F: A0 00 SDA LDY #50 RESULT TO R0
F771: 38 CPR SEC NOTE Y-REG = 13*2 FOR CPR
F772: A5 00 LDA ROL
F774: F5 00 SBC ROL,X
F776: 99 00 00 STA ROL,Y RO-RX TO RY
F779: A5 01 LDA ROH
F77B: F5 01 SBC ROH,X
F77D: 99 00 00 SUD2 STA ROH,Y
F780: 98 TYA LAST RESULT REG*2
F781: 69 00 ADC #50 CARRY TO LSH
F783: 85 1D STA R14H
F785: 60 RTS
F786: A5 00 ADD LDA ROL
F788: 75 00 ADC ROL,X
F78A: 85 00 STA ROL RX+RX TO R0
F78C: A5 01 LDA ROH
F78E: 75 01 ADC ROH,X
F790: A0 00 LDY #50 RO FOR RESULT
F792: 00 E9 BEO SUB2 FINISH ADD
F794: A5 1E BS LDA R15L NOTE X-REG IS 12*21
F796: 20 19 F7 JSR STAT2 PUSH LOW PC BYTE VIA R12
F799: A5 1F LDA R15H
F79B: 20 19 F7 JSR STAT2 PUSH HIGH-ORDER PC BYTE
F79E: 18 BR CLC
F79F: B0 0F BNC ECS BGC2 NO CARRY TEST
F7A1: B1 1F BR1 LDA (R15L),Y DISPLACEMENT BYTE
F7A3: 10 01 SPL R82
F7A5: 68 DEY
F7A6: 65 1E BR2 ADC R15L ADD TO PC
F7A8: 85 1E STA R15H
F7AA: 98 TYA
F7AB: 65 1F ADC R15H
F7AD: 85 1F STA R15H
F7AF: 60 SNC2 RTS
F7B0: 80 EC BC BCS BR
F7B2: 60 RTS
F7B3: 0A 3P ASL \ LOUFE RESULT-REG INDEX
F7B4: AA TAX TO X-REG FOR INDEXING
F7B5: B5 01 LDA R0L,X TEST FOR PLUS
F7B7: 10 E8 EPL RP1 BRANCH IF SO
F7B9: 60 RTS
F7BA: 0A 3H ASL \ DOUBLE RESULT-REG INDEX
F7BB: AA TAX
F7BC: B5 01 LDA R0L,X TEST FOR MINUS
F7BD: 30 E1 BMI RP1
F7C0: 60 RTS
F7C1: 0A BZ ASL \ DOUBLE RESULT-REG INDEX
F7C2: AA TAX
F7C3: B5 00 LDA ROL,X TEST FOR ZERO
F7C5: 15 01 OFA ROL,X (BOTH BYTES)
F7C7: F0 08 DEO RP1 BRANCH IF SO
F7C9: 60 PTS
F7CA: 0A B12 ASL \ DOUBLE RESULT-REG INDEX
F7CB: AA TAX
F7CC: B5 00 LDA ROL,X TEST FOR NONZERO
F7CE: 15 01 ORA ROL,X (BOTH BYTES)
F7DF: D0 CF BNE BR1 BRANCH IF SO
F7D2: 60 PTS
F7D3: 0A BM1 ASL A DOUBLE RESULT-REG INDEX
F7D4: AA TAX
F7D5: B5 00 LDA ROL,X CHECK BOTH BYTES
F7D7: 35 01 AND ROL,X FOR SFF (MINUS 1)
F7D9: 49 FF EOR #SFF
F7DB: F0 C4 BEQ RP1 BRANCH IF SO
F7DD: 60 RIS
F7DE: 0A BM1 ASL A DOUBLE RESULT-REG INDEX
F7DF: AA TAX
F7E0: B5 00 LDA ROL,X CHECK BOTH BYTES FOR NO SFF
F7E2: 35 01 AND ROL,X
F7E4: 49 FF EOR #SFF
F7E6: D0 B9 BNE RP1 BRANCH IF NOT MINUS 1
F7E8: 60 PTS
F7E9: A2 18 RS LDX #S18 12*2 FOR R12 AS STK POINTER
F7EB: 20 66 F7 JSR DCR, DCR STACK POINTER
F7EE: A1 00 LDA (P0L,X) POP HIGH RETURNADR 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
F7FA: 4C C7 F6 RTS
F7FB: 4C C7 F6 RTS
6502 MICROPROCESSOR INSTRUCTIONS

ADC  Add Memory to Accumulator with Carry
AND  "AND" Memory with Accumulator
ASL  Shift Left One Bit (Memory or Accumulator)
BCC  Branch on Carry Clear
BCS  Branch on Carry Set
BEQ  Branch on Result Zero
BIT  Test Bits in Memory with Accumulator
BMI  Branch on Result Minus
BNE  Branch on Result not Zero
BPL  Branch on Result Plus
BRK  Force Break
BVC  Branch on Overflow Clear
BVS  Branch on Overflow Set
CLC  Clear Carry Flag
CLD  Clear Decimal Mode
CLI  Clear Interrupt Disable Bit
CLV  Clear Overflow Flag
CMP  Compare Memory and Accumulator
CPX  Compare Memory and Index X
CPY  Compare Memory and Index Y
DEC  Decrement Memory by One
DEX  Decrement Index X by One
DEY  Decrement Index Y by One
EOR  "Exclusive-Or" Memory with Accumulator
INC  Increment Memory by One
INX  Increment Index X by One
INY  Increment Index Y by One
JMP  Jump to New Location
JSR  Jump to New Location Saving Return Address
LDA  Load Accumulator with Memory
LDX  Load Index X with Memory
LDY  Load Index Y with Memory
LSR  Shift Right one Bit (Memory or Accumulator)
NOP  No Operation
ORA  "OR" Memory with Accumulator
PHA  Push Accumulator on Stack
PHP  Push Processor Status on Stack
PLA  Pull Accumulator from Stack
PLP  Pull Processor Status from Stack
ROL  Rotate One Bit Left (Memory or Accumulator)
ROR  Rotate One Bit Right (Memory or Accumulator)
RTI  Return from Interrupt
RTS  Return from Subroutine
SBC  Subtract Memory from Accumulator with Borrow
SEC  Set Carry Flag
SED  Set Decimal Mode
SEI  Set Interrupt Disable Status
STA  Store Accumulator in Memory
STX  Store Index X in Memory
STY  Store Index Y in Memory
TAX  Transfer Accumulator to Index X
TAY  Transfer Accumulator to Index Y
TXA  Transfer Index X to Accumulator
TXS  Transfer Index X to Stack Pointer
TYA  Transfer Index Y to Accumulator

100
THE FOLLOWING NOTATION APPLIES TO THIS SUMMARY:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Accumulator</td>
</tr>
<tr>
<td>X, Y</td>
<td>Index Registers</td>
</tr>
<tr>
<td>M</td>
<td>Memory</td>
</tr>
<tr>
<td>C</td>
<td>Borrow</td>
</tr>
<tr>
<td>P</td>
<td>Processor Status Register</td>
</tr>
<tr>
<td>S</td>
<td>Stack Pointer</td>
</tr>
<tr>
<td>✓</td>
<td>Change</td>
</tr>
<tr>
<td>—</td>
<td>No Change</td>
</tr>
<tr>
<td>+</td>
<td>Add</td>
</tr>
<tr>
<td>∨</td>
<td>Logical AND</td>
</tr>
<tr>
<td>−</td>
<td>Subtract</td>
</tr>
<tr>
<td>⊕</td>
<td>Logical Exclusive Or</td>
</tr>
<tr>
<td>↑</td>
<td>Transfer From Stack</td>
</tr>
<tr>
<td>↓</td>
<td>Transfer To Stack</td>
</tr>
<tr>
<td>←</td>
<td>Transfer To</td>
</tr>
<tr>
<td>∨</td>
<td>Logical OR</td>
</tr>
<tr>
<td>PC</td>
<td>Program Counter</td>
</tr>
<tr>
<td>PCH</td>
<td>Program Counter High</td>
</tr>
<tr>
<td>PCL</td>
<td>Program Counter Low</td>
</tr>
<tr>
<td>OPER</td>
<td>Operand</td>
</tr>
<tr>
<td>#</td>
<td>Immediate Addressing Mode</td>
</tr>
</tbody>
</table>

FIGURE 1: ASL-SHIFT LEFT ONE BIT OPERATION

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

FIGURE 3: 

Note 1: BIT — TEST BITS

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

PROGRAMMING MODEL

ACCUMULATOR

INDEX REGISTER Y

INDEX REGISTER X

PROGRAM COUNTER

STACK POINTER

PROCESSOR STATUS REGISTER, “P”
### INSTRUCTION CODES

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX DP Code</th>
<th>No. Bytes</th>
<th>&quot;P&quot; Status Reg. N Z C I D V</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ADC</strong></td>
<td>A·M·C → A·C</td>
<td>Immediate</td>
<td>ADC #Oper</td>
<td>69</td>
<td>2</td>
<td>√√√−−−−√</td>
</tr>
<tr>
<td>Add memory to accumulator with carry</td>
<td>A·M·C + A·C</td>
<td>Zero Page, Absolute, (indirect) X, Y</td>
<td>ADC Oper, ADC Oper.X, ADC Oper.Y</td>
<td>62, 70, 79</td>
<td>3, 3, 3</td>
<td></td>
</tr>
<tr>
<td><strong>AND</strong></td>
<td>A·M·A → A</td>
<td>Immediate</td>
<td>AND #Oper</td>
<td>29</td>
<td>2</td>
<td>√√√−−−−√</td>
</tr>
<tr>
<td>&quot;AND&quot; memory with accumulator</td>
<td>A·M·A + A</td>
<td>Zero Page, Absolute, (indirect) X, Y</td>
<td>AND Oper, AND Oper.X, AND Oper.Y</td>
<td>35, 21, 31</td>
<td>2, 2, 2</td>
<td></td>
</tr>
<tr>
<td><strong>ASL</strong></td>
<td>(See Figure 1)</td>
<td>Accumulator</td>
<td>ASL A, ASL Oper, ASL Oper.X</td>
<td>0A, 16, 0E</td>
<td>1, 2, 3</td>
<td>√√√−−−−√</td>
</tr>
<tr>
<td>Shift left one bit (Memory or Accumulator)</td>
<td></td>
<td>Zero Page, Absolute, (indirect) X, Y</td>
<td>ASL Oper.X, ASL Oper.Y</td>
<td>06, 0E</td>
<td>2, 3</td>
<td></td>
</tr>
<tr>
<td><strong>BCC</strong></td>
<td>Branch on carry clear</td>
<td>Branch on C=0</td>
<td>Relative</td>
<td>BCC Oper</td>
<td>90</td>
<td>2</td>
</tr>
<tr>
<td><strong>BCS</strong></td>
<td>Branch on carry set</td>
<td>Branch on C=1</td>
<td>Relative</td>
<td>BCS Oper</td>
<td>80</td>
<td>2</td>
</tr>
<tr>
<td><strong>BEQ</strong></td>
<td>Branch on zero result</td>
<td>Branch on Z=1</td>
<td>Relative</td>
<td>BEQ Oper</td>
<td>80</td>
<td>2</td>
</tr>
<tr>
<td><strong>BIT</strong></td>
<td>Test bits in memory with accumulator</td>
<td>A·M·M, M·Y → N, M·Y → M, Zero Page, Absolute</td>
<td>BIT·Oper, BIT·Oper</td>
<td>24, 25</td>
<td>2, 3</td>
<td>M₉√−−−−√M₉</td>
</tr>
<tr>
<td><strong>BMI</strong></td>
<td>Branch on result minus</td>
<td>Branch on N=1</td>
<td>Relative</td>
<td>BMI Oper</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td><strong>BNE</strong></td>
<td>Branch on result not zero</td>
<td>Branch on Z=0</td>
<td>Relative</td>
<td>BNE Oper</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td><strong>BPL</strong></td>
<td>Branch on result plus</td>
<td>Branch on N=0</td>
<td>Relative</td>
<td>BPL Oper</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td><strong>BRK</strong></td>
<td>Force Break</td>
<td>Forced Interrupt PC=2</td>
<td>BRK· Oper</td>
<td>00</td>
<td>1</td>
<td>---</td>
</tr>
<tr>
<td><strong>BVC</strong></td>
<td>Branch on overflow clear</td>
<td>Branch on V=0</td>
<td>Relative</td>
<td>BVC Oper</td>
<td>50</td>
<td>2</td>
</tr>
</tbody>
</table>

**Note 1:** M₉ < 128 and V < 128.

**Note 2:** A DPL command cannot be executed by setting 1

---

### Additional Instructions

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX DP Code</th>
<th>No. Bytes</th>
<th>&quot;P&quot; Status Reg. N Z C I D V</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BVS</strong></td>
<td>Branch on overflow set</td>
<td>Branch on V=1</td>
<td>Relative</td>
<td>BVS Oper</td>
<td>70</td>
<td>2</td>
</tr>
<tr>
<td><strong>CLC</strong></td>
<td>Clear carry flag</td>
<td>0 → C</td>
<td>Implied</td>
<td>CLC</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td><strong>CLD</strong></td>
<td>Clear decimal mode</td>
<td>0 → D</td>
<td>Implied</td>
<td>CLD</td>
<td>08</td>
<td>1</td>
</tr>
<tr>
<td><strong>CLI</strong></td>
<td>0 → I</td>
<td>Implied</td>
<td>CLI</td>
<td>58</td>
<td>1</td>
<td>0−−−−</td>
</tr>
<tr>
<td><strong>CLV</strong></td>
<td>Clear overflow flag</td>
<td>0 → V</td>
<td>Implied</td>
<td>CLV</td>
<td>88</td>
<td>1</td>
</tr>
<tr>
<td><strong>CMP</strong></td>
<td>Compare memory and accumulator</td>
<td>A → M</td>
<td>Immediate</td>
<td>CMP·Oper</td>
<td>C9</td>
<td>2</td>
</tr>
<tr>
<td>Zero Page, Absolute, (indirect) X, (indirect) Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPX</strong></td>
<td>Compare memory and index X</td>
<td>X → M</td>
<td>Immediate</td>
<td>CPX·Oper</td>
<td>E0</td>
<td>2</td>
</tr>
<tr>
<td>Zero Page, Absolute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPY</strong></td>
<td>Compare memory and index Y</td>
<td>Y → M</td>
<td>Immediate</td>
<td>CPY·Oper</td>
<td>E0</td>
<td>2</td>
</tr>
<tr>
<td>Zero Page, Absolute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DEC</strong></td>
<td>Decrement memory by one</td>
<td>M → M</td>
<td>Decrement</td>
<td>DEC·Oper</td>
<td>C6</td>
<td>2</td>
</tr>
<tr>
<td>Zero Page, Absolute, (indirect) X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DEX</strong></td>
<td>Decrement index X by one</td>
<td>X → X</td>
<td>Implied</td>
<td>DEX·Oper</td>
<td>CA</td>
<td>1</td>
</tr>
<tr>
<td><strong>DEY</strong></td>
<td>Decrement index Y by one</td>
<td>Y → Y</td>
<td>Implied</td>
<td>DEY·Oper</td>
<td>88</td>
<td>1</td>
</tr>
</tbody>
</table>
### INSTRUCTION CODES

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX OP Code</th>
<th>No. Bytes</th>
<th>&quot;P&quot; Status Reg.</th>
<th>N Z C I D V</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EOR</strong></td>
<td>A VM → A</td>
<td>Immediate</td>
<td>EOR #Op</td>
<td>49</td>
<td>2</td>
<td></td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>EOR Op</td>
<td>45</td>
<td>2</td>
<td></td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>EOR Op X</td>
<td>55</td>
<td>2</td>
<td></td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>EOR Op</td>
<td>40</td>
<td>3</td>
<td></td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>EOR Op</td>
<td>50</td>
<td>3</td>
<td></td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>EOR Op Y</td>
<td>59</td>
<td>3</td>
<td></td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect X)</td>
<td>EOR (Op X)</td>
<td>41</td>
<td>2</td>
<td></td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect Y)</td>
<td>EOR (Op Y)</td>
<td>51</td>
<td>2</td>
<td></td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>INC</strong></td>
<td>M + 1 → M</td>
<td>Zero Page</td>
<td>INC Op</td>
<td>E6</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>INC Op X</td>
<td>F6</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>INC Op</td>
<td>EE</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>INC Op</td>
<td>FE</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>INX</strong></td>
<td>X + 1 → X</td>
<td>Implied</td>
<td>INX</td>
<td>E8</td>
<td>1</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>INY</strong></td>
<td>Y + 1 → Y</td>
<td>Implied</td>
<td>INY</td>
<td>C8</td>
<td>1</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>JMP</strong></td>
<td>(PC+1) → PCL</td>
<td>Absolute Indirect</td>
<td>JMP Op</td>
<td>4C</td>
<td>3</td>
<td></td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td>(PC-2) → PCH</td>
<td>Absolute</td>
<td>JMP (Op)</td>
<td>6C</td>
<td>3</td>
<td></td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>JSR</strong></td>
<td>PC+2 → PCH</td>
<td>Absolute</td>
<td>JSR Op</td>
<td>20</td>
<td>3</td>
<td></td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>LDA</strong></td>
<td>M → A</td>
<td>Immediate</td>
<td>LDA #Op</td>
<td>A9</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LDA Op</td>
<td>A5</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>LDA Op X</td>
<td>B5</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>LDA Op</td>
<td>AD</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>LDA Op X</td>
<td>BD</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>LDA (Op X)</td>
<td>B9</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect X)</td>
<td>LDA (Op Y)</td>
<td>A1</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect Y)</td>
<td>LDA (Op Y)</td>
<td>B1</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>LDX</strong></td>
<td>M → X</td>
<td>Immediate</td>
<td>LDX #Op</td>
<td>A2</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LDX Op</td>
<td>A6</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page Y</td>
<td>LDX Op Y</td>
<td>B6</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>LDX Op</td>
<td>AE</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>LDX Op Y</td>
<td>BE</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>LDY</strong></td>
<td>M → Y</td>
<td>Immediate</td>
<td>LDY #Op</td>
<td>A0</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LDY Op</td>
<td>A4</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>LDY Op X</td>
<td>B4</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>LDY Op</td>
<td>AC</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>LDY Op Y</td>
<td>BE</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name Description</th>
<th>Operation</th>
<th>Addressing Mode</th>
<th>Assembly Language Form</th>
<th>HEX OP Code</th>
<th>No. Bytes</th>
<th>&quot;P&quot; Status Reg.</th>
<th>N Z C I D V</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LSR</strong></td>
<td>(See Figure 1)</td>
<td>Accumulator</td>
<td>LSR A</td>
<td>4A</td>
<td>1</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>LSR Op</td>
<td>46</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>LSR Op X</td>
<td>56</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>LSR Op X</td>
<td>4E</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>LSR Op X</td>
<td>5E</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>NOP</strong></td>
<td>No operation</td>
<td>No Operation</td>
<td>NOP</td>
<td>EA</td>
<td>1</td>
<td></td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>ORA</strong></td>
<td>A VM → A</td>
<td>Immediate</td>
<td>ORA #Op</td>
<td>09</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>ORA Op</td>
<td>05</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>ORA Op X</td>
<td>15</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>ORA Op X</td>
<td>00</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>ORA Op X</td>
<td>10</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Y</td>
<td>ORA Op Y</td>
<td>19</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect X)</td>
<td>ORA (Op X)</td>
<td>01</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Indirect Y)</td>
<td>ORA (Op Y)</td>
<td>11</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>PHA</strong></td>
<td>Push accumulator on stack</td>
<td>A ↑</td>
<td>Implied</td>
<td>PHA</td>
<td>48</td>
<td>1</td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>PHP</strong></td>
<td>Push processor status on stack</td>
<td>P ↑</td>
<td>Implied</td>
<td>PHP</td>
<td>08</td>
<td>1</td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>PLA</strong></td>
<td>Pull accumulator from stack</td>
<td>A ↑</td>
<td>Implied</td>
<td>PLA</td>
<td>68</td>
<td>1</td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>PLP</strong></td>
<td>Pull processor status from stack</td>
<td>P ↑</td>
<td>Implied</td>
<td>PLP</td>
<td>28</td>
<td>1</td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>ROL</strong></td>
<td>Rotate one bit left (memory or accumulator)</td>
<td>(See Figure 2)</td>
<td>Accumulator</td>
<td>ROL A</td>
<td>2A</td>
<td>1</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>ROL Op</td>
<td>26</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>ROL Op X</td>
<td>36</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>ROL Op X</td>
<td>25</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>ROL Op X</td>
<td>3E</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td><strong>ROR</strong></td>
<td>Rotate one bit right (memory or accumulator)</td>
<td>(See Figure 3)</td>
<td>Accumulator</td>
<td>ROR A</td>
<td>6A</td>
<td>1</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page</td>
<td>ROR Op</td>
<td>66</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero Page X</td>
<td>ROR Op X</td>
<td>76</td>
<td>2</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>ROR Op X</td>
<td>6E</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute X</td>
<td>ROR Op X</td>
<td>7E</td>
<td>3</td>
<td>(\because)</td>
<td>(\because)</td>
</tr>
<tr>
<td>Name Description</td>
<td>Operation</td>
<td>Addressing Mode</td>
<td>Assembly Language Form</td>
<td>HEX OP Code</td>
<td>No. Bytes</td>
<td>F&quot; Status Reg. N Z C I D V</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
<td>-----------------</td>
<td>------------------------</td>
<td>-------------</td>
<td>-----------</td>
<td>---------------------------</td>
<td></td>
</tr>
<tr>
<td>RTI</td>
<td>P .setValue(PC)</td>
<td>Implied</td>
<td>RTI</td>
<td>40</td>
<td>1</td>
<td>From Stack</td>
<td></td>
</tr>
<tr>
<td>RTS</td>
<td>PC, PC-1, PC</td>
<td>Implied</td>
<td>RTS</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBC</td>
<td>A - M - C A</td>
<td>Immediate</td>
<td>SBC @ Oper</td>
<td>E9</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Zero Page</td>
<td>SBC Oper</td>
<td>E5</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Zero Page X</td>
<td>SBC Oper X</td>
<td>F5</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Absolute</td>
<td>SBC Oper</td>
<td>E0</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Absolute X</td>
<td>SBC Oper X</td>
<td>F0</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Absolute Y</td>
<td>SBC Oper Y</td>
<td>F9</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(indirect X)</td>
<td>SBC (Oper.X)</td>
<td>E1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(indirect Y)</td>
<td>SBC (Oper.Y)</td>
<td>F1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEC</td>
<td>1 C</td>
<td>Implied</td>
<td>SEC</td>
<td>38</td>
<td>1</td>
<td>-1-</td>
<td></td>
</tr>
<tr>
<td>SED</td>
<td>1 D</td>
<td>Implied</td>
<td>SED</td>
<td>F8</td>
<td>1</td>
<td>-1-</td>
<td></td>
</tr>
<tr>
<td>SEI</td>
<td>1 I</td>
<td>Implied</td>
<td>SEI</td>
<td>78</td>
<td>1</td>
<td>-1-</td>
<td></td>
</tr>
<tr>
<td>STA</td>
<td>A M</td>
<td>Zero Page</td>
<td>STA Oper</td>
<td>85</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Zero Page</td>
<td>STA Oper.X</td>
<td>95</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Absolute</td>
<td>STA Oper</td>
<td>80</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Absolute X</td>
<td>STA Oper X</td>
<td>90</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Absolute Y</td>
<td>STA Oper Y</td>
<td>99</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(indirect X)</td>
<td>STA (Oper.X)</td>
<td>81</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(indirect Y)</td>
<td>STA (Oper.Y)</td>
<td>91</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STX</td>
<td>X M</td>
<td>Zero Page</td>
<td>STX Oper</td>
<td>86</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Zero Page X</td>
<td>STX Oper Y</td>
<td>96</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Absolute</td>
<td>STX Oper</td>
<td>8E</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STY</td>
<td>Y M</td>
<td>Zero Page</td>
<td>STY Oper</td>
<td>84</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Zero Page X</td>
<td>STY Oper Y</td>
<td>94</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Absolute</td>
<td>STY Oper</td>
<td>8C</td>
<td>3</td>
<td></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>F&quot; Status Reg. N Z C I D V</th>
</tr>
</thead>
<tbody>
<tr>
<td>TXA</td>
<td>X A</td>
<td>Implied</td>
<td>TXA</td>
<td>8A</td>
<td>1</td>
<td>√√--</td>
</tr>
<tr>
<td>TXS</td>
<td>X S</td>
<td>Implied</td>
<td>TXS</td>
<td>9A</td>
<td>1</td>
<td>--</td>
</tr>
<tr>
<td>TYA</td>
<td>Y A</td>
<td>Implied</td>
<td>TYA</td>
<td>9B</td>
<td>1</td>
<td>√√--</td>
</tr>
</tbody>
</table>
### HEX OPERATION CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>BRK</td>
</tr>
<tr>
<td>01</td>
<td>ORA (Indirect, X)</td>
</tr>
<tr>
<td>02</td>
<td>NOP</td>
</tr>
<tr>
<td>03</td>
<td>NOP</td>
</tr>
<tr>
<td>04</td>
<td>NOP</td>
</tr>
<tr>
<td>05</td>
<td>ORA (Zero Page)</td>
</tr>
<tr>
<td>06</td>
<td>ASL (Zero Page)</td>
</tr>
<tr>
<td>07</td>
<td>NOP</td>
</tr>
<tr>
<td>08</td>
<td>PHP</td>
</tr>
<tr>
<td>09</td>
<td>ORA (Immediate)</td>
</tr>
<tr>
<td>0A</td>
<td>ASL (Accumulator)</td>
</tr>
<tr>
<td>0B</td>
<td>NOP</td>
</tr>
<tr>
<td>0C</td>
<td>NOP</td>
</tr>
<tr>
<td>0D</td>
<td>ORA (Absolute)</td>
</tr>
<tr>
<td>0E</td>
<td>ASL (Absolute)</td>
</tr>
<tr>
<td>0F</td>
<td>NOP</td>
</tr>
<tr>
<td>10</td>
<td>BPL</td>
</tr>
<tr>
<td>11</td>
<td>ORA (Indirect), Y</td>
</tr>
<tr>
<td>12</td>
<td>NOP</td>
</tr>
<tr>
<td>13</td>
<td>NOP</td>
</tr>
<tr>
<td>14</td>
<td>NOP</td>
</tr>
<tr>
<td>15</td>
<td>ORA (Zero Page, X)</td>
</tr>
<tr>
<td>16</td>
<td>ASL (Zero Page, X)</td>
</tr>
<tr>
<td>17</td>
<td>NOP</td>
</tr>
<tr>
<td>18</td>
<td>CLC</td>
</tr>
<tr>
<td>19</td>
<td>ORA (Absolute, Y)</td>
</tr>
<tr>
<td>1A</td>
<td>NOP</td>
</tr>
<tr>
<td>1B</td>
<td>NOP</td>
</tr>
<tr>
<td>1C</td>
<td>NOP</td>
</tr>
<tr>
<td>1D</td>
<td>ORA (Absolute, X)</td>
</tr>
<tr>
<td>1E</td>
<td>ASL (Absolute, X)</td>
</tr>
<tr>
<td>1F</td>
<td>NOP</td>
</tr>
<tr>
<td>20</td>
<td>JSR</td>
</tr>
<tr>
<td>21</td>
<td>AND (Indirect, X)</td>
</tr>
<tr>
<td>22</td>
<td>NOP</td>
</tr>
<tr>
<td>23</td>
<td>NOP</td>
</tr>
<tr>
<td>24</td>
<td>BIT (Zero Page)</td>
</tr>
<tr>
<td>25</td>
<td>AND (Zero Page)</td>
</tr>
<tr>
<td>26</td>
<td>ROL (Zero Page)</td>
</tr>
<tr>
<td>27</td>
<td>NOP</td>
</tr>
<tr>
<td>28</td>
<td>PLP</td>
</tr>
<tr>
<td>29</td>
<td>AND (Immediate)</td>
</tr>
<tr>
<td>2A</td>
<td>ROL (Immediate)</td>
</tr>
<tr>
<td>2B</td>
<td>NOP</td>
</tr>
<tr>
<td>2C</td>
<td>BIT (Absolute)</td>
</tr>
<tr>
<td>2D</td>
<td>AND (Absolute)</td>
</tr>
<tr>
<td>2E</td>
<td>ROL (Absolute)</td>
</tr>
<tr>
<td>2F</td>
<td>NOP</td>
</tr>
<tr>
<td>30</td>
<td>NOP</td>
</tr>
<tr>
<td>31</td>
<td>AND (Indirect, Y)</td>
</tr>
<tr>
<td>32</td>
<td>NOP</td>
</tr>
<tr>
<td>33</td>
<td>NOP</td>
</tr>
<tr>
<td>34</td>
<td>NOP</td>
</tr>
<tr>
<td>35</td>
<td>AND (Zero Page, X)</td>
</tr>
<tr>
<td>36</td>
<td>ROL - Zero Page</td>
</tr>
<tr>
<td>37</td>
<td>NOP</td>
</tr>
<tr>
<td>38</td>
<td>SEC</td>
</tr>
<tr>
<td>39</td>
<td>AND - Absolute, Y</td>
</tr>
<tr>
<td>3A</td>
<td>NOP</td>
</tr>
<tr>
<td>3B</td>
<td>NOP</td>
</tr>
<tr>
<td>3C</td>
<td>NOP</td>
</tr>
<tr>
<td>3D</td>
<td>AND - Absolute, X</td>
</tr>
<tr>
<td>3E</td>
<td>ROL - Absolute, X</td>
</tr>
<tr>
<td>3F</td>
<td>NOP</td>
</tr>
<tr>
<td>40</td>
<td>RTI</td>
</tr>
<tr>
<td>41</td>
<td>EOR (Indirect, X)</td>
</tr>
<tr>
<td>42</td>
<td>NOP</td>
</tr>
<tr>
<td>43</td>
<td>NOP</td>
</tr>
<tr>
<td>44</td>
<td>NOP</td>
</tr>
<tr>
<td>45</td>
<td>EOR - Zero Page</td>
</tr>
<tr>
<td>46</td>
<td>LSR - Zero Page</td>
</tr>
<tr>
<td>47</td>
<td>NOP</td>
</tr>
<tr>
<td>48</td>
<td>PHA</td>
</tr>
<tr>
<td>49</td>
<td>EOR - Immediate</td>
</tr>
<tr>
<td>4A</td>
<td>LSR - Accumulator</td>
</tr>
<tr>
<td>4B</td>
<td>NOP</td>
</tr>
<tr>
<td>4C</td>
<td>JMP - Absolute</td>
</tr>
<tr>
<td>4D</td>
<td>EOR - Absolute</td>
</tr>
<tr>
<td>4E</td>
<td>LSR - Absolute</td>
</tr>
<tr>
<td>4F</td>
<td>NOP</td>
</tr>
<tr>
<td>50</td>
<td>BVC</td>
</tr>
<tr>
<td>51</td>
<td>EOR (Indirect, Y)</td>
</tr>
<tr>
<td>52</td>
<td>NOP</td>
</tr>
<tr>
<td>53</td>
<td>NOP</td>
</tr>
<tr>
<td>54</td>
<td>NOP</td>
</tr>
<tr>
<td>55</td>
<td>EOR - Zero Page</td>
</tr>
<tr>
<td>56</td>
<td>LSR - Zero Page</td>
</tr>
<tr>
<td>57</td>
<td>NOP</td>
</tr>
<tr>
<td>58</td>
<td>LSR - Absolute</td>
</tr>
<tr>
<td>59</td>
<td>EOR - Absolute, Y</td>
</tr>
<tr>
<td>5A</td>
<td>NOP</td>
</tr>
<tr>
<td>5B</td>
<td>NOP</td>
</tr>
<tr>
<td>5C</td>
<td>NOP</td>
</tr>
<tr>
<td>5D</td>
<td>EOR - Absolute, X</td>
</tr>
<tr>
<td>5E</td>
<td>LSR - Absolute, X</td>
</tr>
<tr>
<td>5F</td>
<td>NOP</td>
</tr>
<tr>
<td>60</td>
<td>RTS</td>
</tr>
<tr>
<td>61</td>
<td>ADC - Indirect, X</td>
</tr>
<tr>
<td>62</td>
<td>NOP</td>
</tr>
<tr>
<td>63</td>
<td>NOP</td>
</tr>
<tr>
<td>64</td>
<td>NOP</td>
</tr>
<tr>
<td>65</td>
<td>ADC - Zero Page</td>
</tr>
<tr>
<td>66</td>
<td>ROR - Zero Page</td>
</tr>
<tr>
<td>67</td>
<td>NOP</td>
</tr>
<tr>
<td>68</td>
<td>PLA</td>
</tr>
<tr>
<td>69</td>
<td>ADC - Immediate</td>
</tr>
<tr>
<td>6A</td>
<td>ROR - Accumulator</td>
</tr>
<tr>
<td>6B</td>
<td>NOP</td>
</tr>
<tr>
<td>6C</td>
<td>JMP - Indirect</td>
</tr>
<tr>
<td>6D</td>
<td>ADC - Absolute</td>
</tr>
<tr>
<td>6E</td>
<td>ROR - Absolute</td>
</tr>
<tr>
<td>6F</td>
<td>NOP</td>
</tr>
<tr>
<td>70</td>
<td>BVS</td>
</tr>
<tr>
<td>71</td>
<td>ADC - (Indirect), Y</td>
</tr>
<tr>
<td>72</td>
<td>NOP</td>
</tr>
<tr>
<td>73</td>
<td>NOP</td>
</tr>
<tr>
<td>74</td>
<td>NOP</td>
</tr>
<tr>
<td>75</td>
<td>ADC - Zero Page</td>
</tr>
<tr>
<td>76</td>
<td>ROR - Zero Page</td>
</tr>
<tr>
<td>77</td>
<td>NOP</td>
</tr>
<tr>
<td>78</td>
<td>SEI</td>
</tr>
<tr>
<td>79</td>
<td>ADC - Absolute, Y</td>
</tr>
<tr>
<td>7A</td>
<td>NOP</td>
</tr>
<tr>
<td>7B</td>
<td>NOP</td>
</tr>
<tr>
<td>7C</td>
<td>NOP</td>
</tr>
<tr>
<td>7D</td>
<td>ADC - Absolute, X</td>
</tr>
<tr>
<td>7E</td>
<td>ROR - Absolute, X</td>
</tr>
<tr>
<td>7F</td>
<td>NOP</td>
</tr>
<tr>
<td>80</td>
<td>NOP</td>
</tr>
<tr>
<td>81</td>
<td>STA - (Indirect, X)</td>
</tr>
<tr>
<td>82</td>
<td>NOP</td>
</tr>
<tr>
<td>83</td>
<td>NOP</td>
</tr>
<tr>
<td>84</td>
<td>STY - Zero Page</td>
</tr>
<tr>
<td>85</td>
<td>STA - Zero Page</td>
</tr>
<tr>
<td>86</td>
<td>STX - Zero Page</td>
</tr>
<tr>
<td>87</td>
<td>NOP</td>
</tr>
<tr>
<td>88</td>
<td>DEX</td>
</tr>
<tr>
<td>89</td>
<td>NOP</td>
</tr>
<tr>
<td>8A</td>
<td>TXA</td>
</tr>
<tr>
<td>8B</td>
<td>NOP</td>
</tr>
<tr>
<td>8C</td>
<td>STY - Absolute</td>
</tr>
<tr>
<td>8D</td>
<td>STA - Absolute</td>
</tr>
<tr>
<td>8E</td>
<td>STX - Absolute</td>
</tr>
<tr>
<td>8F</td>
<td>NOP</td>
</tr>
<tr>
<td>90</td>
<td>BCC</td>
</tr>
<tr>
<td>91</td>
<td>STA - (Indirect, Y)</td>
</tr>
<tr>
<td>92</td>
<td>NOP</td>
</tr>
<tr>
<td>93</td>
<td>NOP</td>
</tr>
<tr>
<td>94</td>
<td>STY - Zero Page</td>
</tr>
<tr>
<td>95</td>
<td>STA - Zero Page</td>
</tr>
<tr>
<td>96</td>
<td>STX - Zero Page</td>
</tr>
<tr>
<td>97</td>
<td>NOP</td>
</tr>
<tr>
<td>98</td>
<td>TYA</td>
</tr>
<tr>
<td>99</td>
<td>STA - Absolute, Y</td>
</tr>
<tr>
<td>9A</td>
<td>TXS</td>
</tr>
<tr>
<td>9B</td>
<td>NOP</td>
</tr>
<tr>
<td>9C</td>
<td>NOP</td>
</tr>
<tr>
<td>9D</td>
<td>STA - Absolute, X</td>
</tr>
<tr>
<td>9E</td>
<td>NOP</td>
</tr>
<tr>
<td>9F</td>
<td>NOP</td>
</tr>
<tr>
<td>A0</td>
<td>LDY - Immediate</td>
</tr>
<tr>
<td>A1</td>
<td>LDA - (Indirect, X)</td>
</tr>
<tr>
<td>A2</td>
<td>LDX - Immediate</td>
</tr>
<tr>
<td>A3</td>
<td>NOP</td>
</tr>
<tr>
<td>A4</td>
<td>LDY - Zero Page</td>
</tr>
<tr>
<td>A5</td>
<td>LDA - Zero Page</td>
</tr>
<tr>
<td>A6</td>
<td>LDX - Zero Page</td>
</tr>
<tr>
<td>A7</td>
<td>NOP</td>
</tr>
<tr>
<td>A8</td>
<td>TAY</td>
</tr>
<tr>
<td>A9</td>
<td>LDA - Immediate</td>
</tr>
<tr>
<td>AA</td>
<td>TAX</td>
</tr>
<tr>
<td>AB</td>
<td>NOP</td>
</tr>
<tr>
<td>AC</td>
<td>LDY - Absolute</td>
</tr>
<tr>
<td>AD</td>
<td>Absolute</td>
</tr>
<tr>
<td>AE</td>
<td>LDX - Absolute</td>
</tr>
<tr>
<td>AF</td>
<td>NOP</td>
</tr>
<tr>
<td>B0</td>
<td>BCS</td>
</tr>
<tr>
<td>B1</td>
<td>LDA - (Indirect, Y)</td>
</tr>
<tr>
<td>B2</td>
<td>NOP</td>
</tr>
<tr>
<td>B3</td>
<td>NOP</td>
</tr>
<tr>
<td>B4</td>
<td>LDY - Zero Page</td>
</tr>
<tr>
<td>B5</td>
<td>LDA - Zero Page</td>
</tr>
<tr>
<td>B6</td>
<td>LDX - Zero Page</td>
</tr>
<tr>
<td>B7</td>
<td>NOP</td>
</tr>
<tr>
<td>B8</td>
<td>CLV</td>
</tr>
<tr>
<td>B9</td>
<td>LDA - Absolute, Y</td>
</tr>
<tr>
<td>BA</td>
<td>TSX</td>
</tr>
<tr>
<td>BB</td>
<td>NOP</td>
</tr>
<tr>
<td>BC</td>
<td>LDY - Absolute</td>
</tr>
<tr>
<td>BD</td>
<td>LDA - Absolute</td>
</tr>
<tr>
<td>BE</td>
<td>LDX - Absolute</td>
</tr>
<tr>
<td>BF</td>
<td>NOP</td>
</tr>
<tr>
<td>C0</td>
<td>CPY - Immediate</td>
</tr>
<tr>
<td>C1</td>
<td>CMP - (Indirect, X)</td>
</tr>
<tr>
<td>C2</td>
<td>NOP</td>
</tr>
<tr>
<td>C3</td>
<td>NOP</td>
</tr>
<tr>
<td>C4</td>
<td>CPY - Zero Page</td>
</tr>
<tr>
<td>C5</td>
<td>CMP - Zero Page</td>
</tr>
<tr>
<td>C6</td>
<td>DEC - Zero Page</td>
</tr>
<tr>
<td>C7</td>
<td>NOP</td>
</tr>
<tr>
<td>C8</td>
<td>INY</td>
</tr>
<tr>
<td>C9</td>
<td>CMP - Immediate</td>
</tr>
<tr>
<td>CA</td>
<td>DEX</td>
</tr>
<tr>
<td>CB</td>
<td>NOP</td>
</tr>
<tr>
<td>CC</td>
<td>CPY - Absolute</td>
</tr>
<tr>
<td>CD</td>
<td>CMP - Absolute</td>
</tr>
<tr>
<td>CE</td>
<td>DEC - Absolute</td>
</tr>
<tr>
<td>CF</td>
<td>NOP</td>
</tr>
<tr>
<td>D0</td>
<td>BNE</td>
</tr>
<tr>
<td>D1</td>
<td>CMP - (Indirect), Y</td>
</tr>
<tr>
<td>D2</td>
<td>NOP</td>
</tr>
<tr>
<td>D3</td>
<td>NOP</td>
</tr>
<tr>
<td>D4</td>
<td>NOP</td>
</tr>
<tr>
<td>D5</td>
<td>CMP - Zero Page</td>
</tr>
<tr>
<td>D6</td>
<td>DEC - Zero Page</td>
</tr>
<tr>
<td>D7</td>
<td>NOP</td>
</tr>
<tr>
<td>D8</td>
<td>CLD</td>
</tr>
<tr>
<td>D9</td>
<td>CMP - Absolute, Y</td>
</tr>
<tr>
<td>DA</td>
<td>NOP</td>
</tr>
<tr>
<td>DB</td>
<td>NOP</td>
</tr>
<tr>
<td>DC</td>
<td>NOP</td>
</tr>
<tr>
<td>DD</td>
<td>CMP - Absolute, X</td>
</tr>
<tr>
<td>DE</td>
<td>DEC - Absolute, X</td>
</tr>
<tr>
<td>DF</td>
<td>NOP</td>
</tr>
<tr>
<td>E0</td>
<td>CPX - Immediate</td>
</tr>
<tr>
<td>E1</td>
<td>SBC - (Indirect, X)</td>
</tr>
<tr>
<td>E2</td>
<td>NOP</td>
</tr>
<tr>
<td>E3</td>
<td>NOP</td>
</tr>
<tr>
<td>E4</td>
<td>CPX - Zero Page</td>
</tr>
<tr>
<td>E5</td>
<td>SBC - Zero Page</td>
</tr>
<tr>
<td>E6</td>
<td>INC - Zero Page</td>
</tr>
<tr>
<td>E7</td>
<td>NOP</td>
</tr>
<tr>
<td>E8</td>
<td>INX</td>
</tr>
<tr>
<td>E9</td>
<td>SBC - Immediate</td>
</tr>
<tr>
<td>EA</td>
<td>NOP</td>
</tr>
<tr>
<td>EB</td>
<td>NOP</td>
</tr>
<tr>
<td>EC</td>
<td>CPX - Absolute</td>
</tr>
<tr>
<td>ED</td>
<td>SBC - Absolute</td>
</tr>
<tr>
<td>EE</td>
<td>INC - Absolute</td>
</tr>
<tr>
<td>EF</td>
<td>NOP</td>
</tr>
<tr>
<td>F0</td>
<td>BGD</td>
</tr>
<tr>
<td>F1</td>
<td>SBC - (Indirect, Y)</td>
</tr>
<tr>
<td>F2</td>
<td>NOP</td>
</tr>
<tr>
<td>F3</td>
<td>NOP</td>
</tr>
<tr>
<td>F4</td>
<td>NOP</td>
</tr>
<tr>
<td>F5</td>
<td>SBC - Zero Page</td>
</tr>
<tr>
<td>F6</td>
<td>INC - Zero Page</td>
</tr>
<tr>
<td>F7</td>
<td>NOP</td>
</tr>
<tr>
<td>F8</td>
<td>SED</td>
</tr>
<tr>
<td>F9</td>
<td>SBC - Absolute</td>
</tr>
<tr>
<td>FA</td>
<td>NOP</td>
</tr>
<tr>
<td>FB</td>
<td>NOP</td>
</tr>
<tr>
<td>FC</td>
<td>NOP</td>
</tr>
<tr>
<td>FD</td>
<td>SBC - Absolute</td>
</tr>
<tr>
<td>FE</td>
<td>INC - Absolute</td>
</tr>
<tr>
<td>FF</td>
<td>NOP</td>
</tr>
</tbody>
</table>
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:

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

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

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

2. +5 Volts at 1.6 amps

3. -5 Volts at 10mA.

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

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

k. Cable for the following:
   1. Keyboard to APPLE II P.C.B.
   2. Video out 75 ohm cable to TV or modulator
   3. Cassette to APPLE II P.C.B. (1 or 2)

Optionally you may desire:

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

m. Case to hold all the above

Final Assembly Steps

1. Using detailed information on pin functions in hardware section of manual, connect power supplies to d.c. cable assembly. Use both ground wires to minimize resistance. With cable assembly disconnected from APPLE II 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 "@" (shift-P) to clear screen. You may now try "Monitor" commands if you wish. See details in "Monitor" software section.

RUNNING BASIC

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

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

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

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

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

Regulation of the DC voltages is accomplished by controlling the frequency at which the converter operates; the greater the output power needed, the lower the frequency of the converter. If the converter is overloaded, the operating frequency will drop into the audible range with squeals 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., 197V 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 discon-
nected, 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 1/2 that of the +5V.
The -5V supply load is 1/10 that of the +5V. The
-12V supply load is 1/10 that of the +5V.

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

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

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

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

CAUTION: There are dangerous high
voltages inside the power supply
case. Much of the internal circuitry
is NOT isolated from the power line,
and special equipment is needed for
service. NO REPAIR BY THE USER IS
ALLOWED.
NOTES ON INTERFACING WITH THE HOME TV

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

Users report success with the following RF modulators:

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

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

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

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

Most of the above are available through local computer stores.

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

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

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

Some cautions are in order. The Apple II, by virtue of its color graphics capability, operates the TV set in a linear mode rather than the 100% contrast mode satisfactory for displaying text. For this reason, radio frequency interference (RFI) generated by a computer (or peripherals) will beat with the
carrier of the RF modulator to produce faint spurious background patterns (called "worms") This RFI "trash" must be of quite a low level if worms are to be prevented. In fact, these spurious beats must be 40 to 50db below the signal level to reduce worms to an acceptable level. When it is remembered that only 2 to 6 mV (across 300Ω) is presented to the VHF input of the TV set, then stray RFI getting into the TV must be less than 50μV to obtain a clean picture. Therefore we recommend that a good, co-ax cable be used to carry the signal from any modulator to the TV set, such as RG/59u (with copper shield), Belden #8241 or an equivalent miniature type such as Belden #8218. We also recommend that the RF modulator be enclosed 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 auxilliary video connector found on the Apple II board. This kit has order number A2M010X. The M&R "Sup-r-Mod" is supplied with a coax cable and toroids.

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

Another caution concerns possible long term effects on the TV picture tube. Most home TV sets have "Brightness" and "Contrast" controls with a very wide range of adjustment. When an un-changing picture is displayed with high brightness for a long period, a faint discoloration of the TV CRT may occur as an inverse pattern observable with the TV set turned off. This condition may be avoided by keeping the "Brightness" turned down slightly and "Contrast" moderate.
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 1a) It can be set up for a 20mA current loop and interfaced as follows (whether or not the teletype is strapped for full duplex or half duplex operation):

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

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

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

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

A Serial Out Machine Center Language Program

Once the appropriate circuit has been selected and constructed a machine language program is needed to drive the circuit. Figure 3 lists such a teletype output machine language routine. It can be used in conjunction with an Integer BASIC program that doesn't require page $300 hex of memory. This program resides in memory from $370 to $3E9. Columns three and four of the listing show the op-code used. To enter this program into the Apple II the following procedure is followed:

Entering Machine Language Program

1. Power up Apple II
2. Depress and release the "RESET" key. An asterick and flashing cursor should appear on the left hand side of the screen below the random text matrix.
3. Now type in the data from columns one, two and three for each line from $370 to $3E9. For example, type in "370: A9 82" and then depress and release the "RETURN" key. Then repeat this procedure for the data at $372 and on until you complete entering the program.

Executing this Program

1. From BASIC a CALL 880 ($370) will start the execution of this program. It will use the teletype or suitable 80 column printer as the primary output device.
2. PR#0 will inactivate the printer transferring control back to the Video monitor as the primary output device.

3. In Monitor mode $370G 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#0
25 END
```

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

Conclusion

With the circuits and machine language program described in this paper the user may develop a relatively simple serial output interface to an ASR-33 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 1  ASR-33

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

TITLE 'TELETEYPE DRIVER ROUTINES'

****************************************************
* TTYDRIVER: *
* TELETEYPE OUTPUT *
* ROUTINE FOR 72 *
* COLUMN PRINT WITH *
* BASIC LIST *
* *
* COPYRIGHT 1977 BY: *
* APPLE COMPUTER INC. *
* 11/18/77 *
* *
* R. WIGGINTON *
* S. WOZNIAK *
* *
****************************************************

WNDWIDTH EQU $21 ; FOR APPLE-II
CH EQU $24 ; CURSOR HORIZ.
CSWL EQU $36 ; CHAR. OUT SWITCH
YSAVE EQU $778 ; COLUMN COUNT LOC.
COLCNT EQU $7F8 ;TO NUMBER COLUMNS ONT
MARK EQU $C058 ;WHERE WE ARE NOW.
SPACE EQU $C059
WAIT EQU $FCA8
ORG $370

***WARNING: OPERAND OVERFLOW IN LINE 27

LDA #TTOUT ;POINT TO TTY ROUTINES
STA CSWL ;HIGH BYTE
LDA #TTOUT/256 ;SET WINDOW WIDTH
STA CSWL+1 ;TO NUMBER COLUMNS ONT
LDA #72 ;TO NUMBER COLUMNS ONT
LDA WNDWIDTH ;WHERE WE ARE NOW.
STA COLCNT

TTINIT: LDA #TTOUT

TTOUT: STA CSWL

TTOUT: LDA COLCNT ;SAVE TWICE

TTOUT2: CMP CH ;ON STACK.

TTOUT2: LDA COLCNT ;CHECK FOR A TAB.

TESTCTRL: BIT RTS1 ;RESTORE OUTPUT CHAR.

TESTCTRL: BEQ PRNTIT ;IF C SET, NO TAB

TESTCTRL: INC COLCNT ;PRINT THE CHAR ON TTY

TESTCTRL: JSR DOCHAR ;RESTORE CHAR

PRNTIT: PLA ;AND PUT BACK ON STACK

PRNTIT: PHA ;DO MORE SPACES FOR TTY

PRNTIT: BCC TTOUT2 ;CHECK FOR CR. RET.

PRNTIT: EOR #$0D

PRNTIT: A SL A ;SELIM PARITY

FINISH BNE FINISH ;IF NOT CR, DONE.

FIGURE 3a
**TELETYPE DRIVER ROUTINES**

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

03A3: 8D F8 07 54  STA COLCNT  CLEAR COLUMN COUNT
03A6: A9 8A 55  LDA #$8A  NOW DO LINE FEED
03A8: 20 C1 03 56  JSR DOCHAR  200MSEC DELAY FOR LINE
03A9: A9 58 57  LDA #$58  CHECK IF IN MARGIN
03AD: 85 24 65  STA CH  FOR CR, RESET CH
03AF: 68 66 68  STA CH  IF SO, CARRY SET.
03B0: AD F8 07 59  FINISH: STA COLCNT
03B3: F0 08 60  JSR DOCHAR
03B5: E5 21 61  JSR WAIT
03B7: E9 F7 62  JSR WAIT
03B9: 90 04 63  LDA VMDWDTH
03BA: 69 1F 64  5B RETURN
03BD: 85 24 65  SETCH: STA CH
03C0: 60 67 68  RETURN: PLA
03C1: 8C 78 07 69  * HERE IS THE TELETYPE PRINT A CHARACTER ROUTINE:
03C4: 08 70  DOCHAR: STY YSAVE  ;RETURN TO CALLER
03C5: A0 03 71  PHP  ;SUCCESSFUL ASSEMBLY: NO ERRORS
03C7: 18 72  LDA #$03  ;CLEAR COLUMN COUNT
03C8: 80 05 74  JSR DOCHAR
03CB: AD 59 C0 75  JSR DOCHAR
03CE: 90 03 76  JSR DOCHAR
03D0: AD 58 C0 77  JSR DOCHAR
03D3: A9 D7 78  JSR DOCHAR
03D5: 4B 79 79  JSR DOCHAR
03D6: A9 20 80  JSR DOCHAR
03D8: 4A 81 81  JSR DOCHAR
03D9: 90 FD 82  JSR DOCHAR
03DB: 6B 83 83  JSR DOCHAR
03DC: E9 01 84  JSR DOCHAR
03DE: D0 F5 85  JSR DOCHAR
03E0: 6B 86 86  JSR DOCHAR
03E1: 6A 37 37  JSR DOCHAR
03E2: 88 88 88  JSR DOCHAR
03E3: D0 E3 39  JSR DOCHAR
03E5: AC 78 07 90  JSR DOCHAR
03E8: 28 91 91  JSR DOCHAR
03E9: 60 92 92  JSR DOCHAR

**FIGURE 3b**
FIGURE 3c
INTERFACING THE APPLE

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

TABLE OF CONTENTS

1. CONNECTOR LOCATION DIAGRAM
2. CASSETTE DATA JACKS (2 EACH)
3. GAME I/O CONNECTOR
4. KEYBOARD CONNECTOR
5. PERIPHERAL CONNECTORS (8 EACH)
6. POWER CONNECTOR
7. SPEAKER CONNECTOR
8. VIDEO OUTPUT JACK
9. AUXILIARY VIDEO OUTPUT CONNECTOR
Figure 18 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. \( V_{IN} = 1 \text{Vpp (nominal)}, Z_{IN} = 12 \text{KOhms.} \) Located at K12 as illustrated in Figure 1.

CASSETTE DATA OUT JACK: Designed for connection to the "MIC" or "MICROPHONE" input found on most audio cassette tape recorders. \( V_{OUT} = 25 \text{mV into 100 Ohms}, Z_{OUT} = 100 \text{Ohms.} \) 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

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

**CQ4Q STB:** A utility strobe output. Will go low during \( \frac{1}{2} \) of a read or write cycle to addresses CQ4Q-CQ4F. This is a 74LSxx series TTL output.

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

**NC:** No connection.

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

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

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

### KEYBOARD CONNECTOR

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

---

**Figure 3**  
**KEYBOARD CONNECTOR**  

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

**LOCATION A7**
SIGNAL DESCRIPTION FOR KEYBOARD INTERFACE

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

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

NC: No connection.

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

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

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

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

PERIPHERAL CONNECTORS

The eight Peripheral Connectors mounted near the back edge of the APPLE II board provide a convenient means of connecting expansion hardware and peripheral devices to the APPLE II I/O Bus. These are Winchester #2HW25C0-111 (or equivalent) 50 pin card edge connectors with pins on .10" centers. Location and pin outs are illustrated in Figures 1 and 4.

SIGNAL DESCRIPTION FOR PERIPHERAL I/O

A0-A15: 16 bit system address bus. Addresses are set up by the 6502 within 300nS after the beginning of Ø1. These lines will drive up to a total of 16 standard TTL loads.

DEVICE SELECT: Sixteen addresses are set aside for each peripheral connector. A read or write to such an address will send pin 41 on the selected connector low during Ø2 (500nS). Each will drive 4 standard TTL loads.

D0-D7: 8 bit system data bus. During a write cycle data is set up by the 6502 less than 300nS after the beginning of Ø2. During a read cycle the 6502 expects data to be ready no less than 100nS before the end of Ø2. These lines will drive up to a total of 8 total low power schottky TTL loads.
DMA: Direct Memory Access control output. This line has a 3K Ohm pullup to +5V and should be driven with an open collector output.

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

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

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

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

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

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

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

I/O STROBE: Pin 20 on all peripheral connectors will go low during Ø2 of a read or write to any address C800-CFFF. This line will drive a total of 4 standard TTL loads.

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

NC: No connection.

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

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

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

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

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

Ø0: Microprocessor phase Ø 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
PINOUT (Back Edge of PC Board)

GND 26 25 +5V
DMA IN 27 24 DMA OUT
INT IN 28 23 INT OUT
NMI 29 22 DMA
IRQ 30 21 RDY
RES 31 20 I/O STROBE
INH 32 19 N.C.
-12V 33 18 R/W
-5V 34 17 A15
N.C. 35 16 A14
7M 36 15 A13
Q3 37 14 A12
+1 38 13 A11
USER 1 39 12 A10
40 40 11 A9
DEVICE SELECT 41 10 A8
D7 42 9 A7
D6 43 8 A6
D5 44 7 A5
D4 45 6 A4
D3 46 5 A3
D2 47 4 A2
D1 48 3 A1
D0 49 2 A0
+12V 50 1 I/O SELECT

(Toward Front Edge of PC Board)
LOCATIONS J2 TO J12

Figure 5  POWER CONNECTOR

TOP VIEW
PINOUT (Toward Right Side of PC Board)

(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.

Figure 6

SPEAKER CONNECTOR

+5V  SPKR

Front Edge of PC Board

Right Edge of PC Board

LOCATION B14A

VIDEO OUTPUT JACK

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

A video level control near the connector allows the output level to be adjusted from 0 to 1 Volt (peak) into an external 75 OHM load.

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

See locations illustrated in Figure 1.
AUXILIARY VIDEO OUTPUT CONNECTOR

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

SIGNAL DESCRIPTION

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

VIDEO: NTSC compatible positive composite VIDEO. DC coupled emitter follower output (not short circuit protected). SYNC TIP is Ø Volts, black level is about .75 Volts, and white level is about 2.0 Volts into 47Ø Ohms. Output level is non-adjustable.

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

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

Figure 7

AUXILIARY VIDEO OUTPUT CONNECTOR

PINOUT

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

Right Edge of PC Board

LOCATION J14B
INSTALLING YOUR OWN RAM

THE POSSIBILITIES

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

RESTRICTIONS

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

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

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

ASSIGNING RAM

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

Figure 8

MEMORY SELECT SOCKETS

TOP VIEW

PINOUT
(0000-0FFF) 4K "0" BLOCK 1 14 RAM ROW C
(1000-1FFF) 4K "1" BLOCK 2 13 RAM ROW D
(2000-3FFF) 4K "2" BLOCK 3 12 RAM ROW E
(3000-4FFF) 4K "3" BLOCK 4 11 N.C.
(4000-5FFF) 4K "4" BLOCK 5 10 16K "0" BLOCK (0000-3FFF)
(5000-6FFF) 4K "5" BLOCK 6 9 16K "4" BLOCK (4000-7FFF)
(6000-7FFF) 4K "6" BLOCK 7 8 16K "8" BLOCK (8000-BFFF)

LOCATIONS D1, E1, F1
MEMORY

TABLE OF CONTENTS

1. INTRODUCTION
2. INSTALLING YOUR OWN RAM
3. MEMORY SELECT SOCKETS
4. MEMORY MAP BY 4K BLOCKS
5. DETAILED MAP OF ASSIGNED ADDRESSES

INTRODUCTION

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

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

A set of three each of the above is supplied with the board. Type 1 is supplied with 4K or 8K systems. Both type 1 and 2 are supplied with 12K systems. Type 1 is a contiguous memory range for maximum BASIC program size. Type 2 is non-contiguous and allows 8K dedicated to HIRES screen memory with approximately 2K of user BASIC space. Type 3 is supplied with 16K, 20K and 24K systems. Type 4 with 30K and 36K systems and type 5 with 48K systems.

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

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

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

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

HOW TO USE

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

Let us learn by example:

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

If in addition you have inserted 4K RAMs into rows "D" and "E", and you want them each to occupy the first 4K of 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>Address</th>
<th>Description</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>text and color graphics display pages, 6502 stack, pointers, etc.</td>
<td>8000</td>
<td>addresses dedicated to hardware functions</td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td>9000</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>high res graphics display primary page</td>
<td>A000</td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td></td>
<td>B000</td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>high res. graphics display secondary page</td>
<td>C000</td>
<td>ROM socket D0: spare</td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td>D000</td>
<td>ROM socket D8: spare</td>
</tr>
<tr>
<td>6000</td>
<td></td>
<td>E000</td>
<td>ROM socket E0: BASIC</td>
</tr>
<tr>
<td>7000</td>
<td></td>
<td>F000</td>
<td>ROM socket FO: BASIC utility</td>
</tr>
</tbody>
</table>

Memory Map Pages Ø 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>UTILIY</td>
<td>register area for &quot;sweet 16&quot; 16 bit firmware processor.</td>
<td></td>
</tr>
<tr>
<td>0000-00FF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0020-004D</td>
<td>MONITOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>004F-004F</td>
<td>MONITOR</td>
<td>holds a 16 bit number that is randomized with each key entry.</td>
<td></td>
</tr>
<tr>
<td>0050-0055</td>
<td>UTILIY</td>
<td>integer multiply and divide work space.</td>
<td></td>
</tr>
<tr>
<td>0055-00FF</td>
<td>BASIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00F0-00FF</td>
<td>UTILIY</td>
<td>floating point work space.</td>
<td></td>
</tr>
<tr>
<td>PAGE ONE</td>
<td>6502</td>
<td>subroutine return stack.</td>
<td></td>
</tr>
<tr>
<td>0100-01FF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAGE TWO</td>
<td>0200-02FF</td>
<td>character input buffer.</td>
<td></td>
</tr>
<tr>
<td>PAGE THREE</td>
<td>MONITOR</td>
<td>Y (control Y) will cause a JSR to this location.</td>
<td></td>
</tr>
<tr>
<td>03F8</td>
<td></td>
<td>NWI's are vectored to this location.</td>
<td></td>
</tr>
<tr>
<td>03FB</td>
<td></td>
<td>IRQ's are vectored to the address pointed to by these locations.</td>
<td></td>
</tr>
<tr>
<td>03FE-03FF</td>
<td>DISPLAY</td>
<td>text or color graphics primary page.</td>
<td></td>
</tr>
<tr>
<td>0400-07FF</td>
<td>DISPLAY</td>
<td>text or color graphics secondary page.</td>
<td></td>
</tr>
<tr>
<td>0800-0BFF</td>
<td>DISPLAY</td>
<td>text or color graphics secondary page.</td>
<td></td>
</tr>
</tbody>
</table>

BASIC initializes LOMEM to location 0800.
### I/O and ROM Address Detail

<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></td>
</tr>
<tr>
<td>C01X</td>
<td>Clear keyboard strobe.</td>
<td></td>
</tr>
<tr>
<td>C02X</td>
<td>Toggle cassette output.</td>
<td></td>
</tr>
<tr>
<td>C03X</td>
<td>Toggle speaker output.</td>
<td></td>
</tr>
<tr>
<td>C04X</td>
<td>&quot;C040 STB&quot;</td>
<td>Output strobe to Game I/O connector.</td>
</tr>
<tr>
<td>C050</td>
<td>Set graphics mode</td>
<td></td>
</tr>
<tr>
<td>C051</td>
<td>&quot; text &quot;</td>
<td></td>
</tr>
<tr>
<td>C052</td>
<td>Set bottom 4 lines graphics</td>
<td></td>
</tr>
<tr>
<td>C053</td>
<td>&quot; &quot; &quot; &quot; text</td>
<td></td>
</tr>
<tr>
<td>C054</td>
<td>Display primary page</td>
<td></td>
</tr>
<tr>
<td>C055</td>
<td>&quot; secondary page</td>
<td></td>
</tr>
<tr>
<td>C056</td>
<td>Set high res. graphics</td>
<td></td>
</tr>
<tr>
<td>C057</td>
<td>&quot; color &quot;</td>
<td></td>
</tr>
<tr>
<td>C058</td>
<td>Clear &quot;AN0&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>
</tbody>
</table>

Keyboard strobe appears in bit 7. ASCII data from keyboard appears in the 7 lower bits.
<table>
<thead>
<tr>
<th>HEX ADDRESS</th>
<th>ASSIGNED FUNCTION</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C060/8</td>
<td>Cassette input</td>
<td>State of &quot;Cassette Data In&quot; appears in bit 7.</td>
</tr>
<tr>
<td>C061/9</td>
<td>&quot;SW1&quot;</td>
<td>State of Switch 1 ^ Game I/O connector appears in bit 7.</td>
</tr>
<tr>
<td>C062/A</td>
<td>&quot;SW2&quot;</td>
<td>State of Switch 2 input on Game I/O connector appears in bit 7.</td>
</tr>
<tr>
<td>C063/B</td>
<td>&quot;SW3&quot;</td>
<td>State of Switch 3 input on Game I/O connector appears in bit 7.</td>
</tr>
<tr>
<td>C064/C</td>
<td>Paddle 0 timer output</td>
<td>State of timer output for Paddle 0 appears in bit 7.</td>
</tr>
<tr>
<td>C065/D</td>
<td>&quot; 1 &quot;</td>
<td>State of timer output for Paddle 1 appears in bit 7.</td>
</tr>
<tr>
<td>C066/E</td>
<td>&quot; 2 &quot;</td>
<td>State of timer output for Paddle 2 appears in bit 7.</td>
</tr>
<tr>
<td>C067/F</td>
<td>&quot; 3 &quot;</td>
<td>State of timer output for Paddle 3 appears in bit 7.</td>
</tr>
<tr>
<td>C07X</td>
<td>&quot;PDL STB&quot;</td>
<td>Triggers paddle timers during $\phi_2$.</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>C0AX</td>
<td>&quot; 2 &quot;</td>
<td></td>
</tr>
<tr>
<td>C0BX</td>
<td>&quot; 3 &quot;</td>
<td></td>
</tr>
<tr>
<td>C0CX</td>
<td>&quot; 4 &quot;</td>
<td></td>
</tr>
<tr>
<td>C0DX</td>
<td>&quot; 5 &quot;</td>
<td></td>
</tr>
<tr>
<td>C0EX</td>
<td>&quot; 6 &quot;</td>
<td></td>
</tr>
<tr>
<td>C0FX</td>
<td>&quot; 7 &quot;</td>
<td></td>
</tr>
<tr>
<td>C10X</td>
<td>&quot; 8 &quot;</td>
<td>Expansion connectors.</td>
</tr>
<tr>
<td>C11X</td>
<td>&quot; 9 &quot;</td>
<td></td>
</tr>
<tr>
<td>C12X</td>
<td>&quot; A &quot;</td>
<td></td>
</tr>
<tr>
<td>HEX ADDRESS</td>
<td>ASSIGNED FUNCTION</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>---------</td>
</tr>
<tr>
<td>C13X</td>
<td>DEVICE SELECT</td>
<td>B</td>
</tr>
<tr>
<td>C14X</td>
<td>&quot;</td>
<td>C</td>
</tr>
<tr>
<td>C15X</td>
<td>&quot;</td>
<td>D</td>
</tr>
<tr>
<td>C16X</td>
<td>&quot;</td>
<td>E</td>
</tr>
<tr>
<td>C17X</td>
<td>&quot;</td>
<td>F</td>
</tr>
<tr>
<td>C1XX</td>
<td>I/O SELECT</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2XX</td>
<td>&quot;</td>
<td>2</td>
</tr>
<tr>
<td>C3XX</td>
<td>&quot;</td>
<td>3</td>
</tr>
<tr>
<td>C4XX</td>
<td>&quot;</td>
<td>4</td>
</tr>
<tr>
<td>C5XX</td>
<td>&quot;</td>
<td>5</td>
</tr>
<tr>
<td>C6XX</td>
<td>&quot;</td>
<td>6</td>
</tr>
<tr>
<td>C7XX</td>
<td>&quot;</td>
<td>7</td>
</tr>
<tr>
<td>C8XX</td>
<td>&quot;</td>
<td>8, I/O STROBE</td>
</tr>
<tr>
<td>C9XX</td>
<td>&quot;</td>
<td>9, &quot;</td>
</tr>
<tr>
<td>CAXX</td>
<td>&quot;</td>
<td>A, &quot;</td>
</tr>
<tr>
<td>CBXX</td>
<td>&quot;</td>
<td>B, &quot;</td>
</tr>
<tr>
<td>CCXX</td>
<td>&quot;</td>
<td>C, &quot;</td>
</tr>
<tr>
<td>CDXX</td>
<td>&quot;</td>
<td>D, &quot;</td>
</tr>
<tr>
<td>CEXX</td>
<td>&quot;</td>
<td>E, &quot;</td>
</tr>
<tr>
<td>CFXX</td>
<td>&quot;</td>
<td>F, &quot;</td>
</tr>
<tr>
<td>D000-D7FF</td>
<td>ROM socket D0</td>
<td></td>
</tr>
<tr>
<td>D800-DFFF</td>
<td>&quot;</td>
<td>D8</td>
</tr>
<tr>
<td>E000-E7FF</td>
<td>&quot;</td>
<td>E0</td>
</tr>
<tr>
<td>E800-EFFF</td>
<td>&quot;</td>
<td>E8</td>
</tr>
<tr>
<td>F000-F7FF</td>
<td>&quot;</td>
<td>F0</td>
</tr>
<tr>
<td>F800-FFFF</td>
<td>&quot;</td>
<td>F8</td>
</tr>
</tbody>
</table>

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

Expansion connectors.

Pin 1 on the selected Peripheral Connector goes low during \( \phi_2 \).

1K of BASIC, 1K of utility.

Monitor.
SYSTEM TIMING

SIGNAL DESCRIPTIONS

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

7M: Intermediate timing signal, 7.159 MHz.

COLOR REF: Color reference frequency used by video circuitry, 3.580 MHz.

\( \Phi_0 \): Phase \( \Phi \) clock to microprocessor, 1.023 MHz nominal.

\( \Phi_1 \): Microprocessor phase 1 clock, complement of \( \Phi_0 \), 1.023 MHz nominal.

\( \Phi_2 \): Same as \( \Phi_0 \). Included here because the 6502 hardware and programming manuals use the designation \( \Phi_2 \) instead of \( \Phi_0 \).

\( \Phi_3 \): 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 \( \Phi_1 \), and is stable about 300nS after the start of \( \Phi_1 \).

DATA WRITE: During a write cycle, data from the microprocessor appears on the data bus during \( \Phi_2 \), and is stable about 300nS after the start of \( \Phi_2 \).

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 \( \Phi_2 \).

SYSTEM TIMING DIAGRAM
FIGURE S-1  APPLE II SYSTEM DIAGRAM
FIGURE S-2 MPU AND SYSTEM BUS
FIGURE S-4  SYNC COUNTER
FIGURE S-6 4K/16K RAM SELECT
FIGURE S-7  RAM ADDRESS MUX
FROM 4K/16K SELECT

FIG. S-6

C3 RAM
C4 RAM
C5 RAM
C6 RAM
C7 RAM
C8 RAM
C9 RAM
C10 RAM
D3 RAM
D4 RAM
D5 RAM
D6 RAM
D7 RAM
D8 RAM
D9 RAM
D10 RAM
E3 RAM
E4 RAM
E5 RAM
E6 RAM
E7 RAM
E8 RAM
E9 RAM
E10 RAM

FROM RAM ADDRESS MUX
FIG. S-7

E1-7 RAM
E12-7 RAM
E13-7 RAM
E11-9 RAM
E12-7 RAM
E13-7 RAM
R/W
R/W
RAS

FIGURE S-8 4K TO 48K RAM MEMORY WITH DATA LATCH
FIGURE S-9  PERIPHERAL I/O CONNECTOR PINOUT AND CONTROL LOGIC
FIGURE S-11 VIDEO GENERATOR