PROGRAMMER'S AID #1

INSTALLATION AND OPERATING MANUAL

TM

Apple Utility Programs

Published by
APPLE COMPUTER INC.
1Ø26Ø Bandley Drive
Cupertino, California 95Ø14
(4Ø8) 996-1Ø1Ø

Reorder APPLE Product #A2LØØ11
(Ø3Ø-ØØ26)

TABLE OF CONTENTS

INTRODUCTION

CHAPTER 1

RENUMBER

CHAPTER 2

APPEND

CHAPTER 3

TAPE VERIFY (BASIC)

CHAPTER 4

TAPE VERIFY (Machine Code or Data)

CHAPTER 5

RELOCATE

CHAPTER 6

RAM TEST

CHAPTER 7

MUSIC

CHAPTER 8

HIGH-RESOLUTION GRAPHICS

APPENDIX II

SOURCE ASSEMBLY LISTINGS

APPENDIX I

SUMMARY OF PROGRAMMER'S AID COMMANDS

INTRODUCTION

FEATURES OF PROGRAMMER'S AID #1

Programmer's Aid #1 combines several APPLE II programs that Integer BASIC programmers need quite frequently. To avoid having to load them from a cassette tape or diskette each time they are used, these programs have been combined in a special read-only memory (ROM) integrated circuit (IC). When this circuit is plugged into one of the empty sockets left on the APPLE's printed-circuit board for this purpose, these programs become a built-in part of the computer the same way Integer BASIC and the Monitor routines are built in. Programmer's Aid #1 allows you to do the following, on your APPLE II:

Chapter 1. Renumber an entire Integer BASIC program, or a portion of the program.

Chapter 2. Load an Integer BASIC program from tape without erasing the Integer BASIC program that was already in memory, in order to combine the two programs.

Chapter 3. Verify that an Integer BASIC program has been saved correctly on tape, before the program is deleted from APPLE's memory.

Chapter 4. Verify that a machine-language program or data area has been saved correctly on tape from the Monitor.

Chapter 5. Relocate 65Ø2 machine-language programs.

Chapter 6. Test the memory of the APPLE.

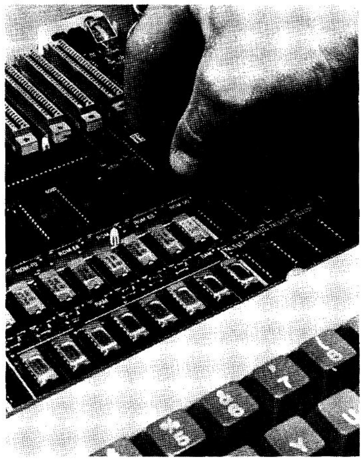
Chapter 7. Generate musical notes of variable duration over four chromatic octaves, in five (slightly) different timbres, from Integer BASIC.

Chapter 8. Do convenient High-Resolution graphics from Integer BASIC.

Note: if your APPLE has the firmware APPLESOFT card installed, its switch must be down (in the Integer BASIC position) for Programmer's Aid #1 to operate.

HOW TO INSTALL THE PROGRAMMER'S AID ROM

The Programmer's Aid ROM is an IC that has to be plugged into a socket on the inside of the APPLE II computer.



1. Turn off the power switch on the back of the APPLE II. This is important to prevent damage to the computer.

2. Remove the cover from the APPLE II. This is done by pulling up on the cover at the rear edge until the two corner fasteners pop apart. Do not continue to lift the rear edge, but slide cover backward until it comes free.

3. Inside the APPLE, toward the right center of the main printed-circuit board, locate the large empty socket in Row F, marked "ROM-DØ".

4. Make sure that the Programmer's Aid ROM IC is oriented correctly. The small semicircular notch should be toward the keyboard. The Programmer's Aid ROM IC must match the orientation of the other ROM ICs that are already installed in that row.

5. Align all the pins on the Programmer's Aid ROM IC with the holes in socket DØ, and gently press the IC into place. If a pin bends, remove the IC from its socket using an "IC puller" (or, less optimally, by prying up gently with a screwdriver). Do not attempt to pull the socket off the board. Straighten any bent pins with a needlenose pliers, and press the IC into its socket again, even more carefully.

6. Replace the cover of the APPLE, remembering to start by sliding the front edge of the cover into position. Press down on the two rear corners until they pop into place.

7. Programmer's Aid #1 is installed; the APPLE II may now be turned on.

CHAPTER 1
RENUMBER

RENUMBERING AN ENTIRE BASIC PROGRAM

After loading your program into the APPLE, type the

```
CLR
```

command. This clears the BASIC variable table, so that the Renumber feature's parameters will be the first variables in the table. The Renumber feature looks for its parameters by location in the variable table. For the parameters to appear in the table in their correct locations, they must be specified in the correct order and they must have names of the correct length.

Now, choose the number you wish assigned to the first line in your renumbered program. Suppose you want your renumbered program to start at line number 1ØØØ. Type

```
START = 1ØØØ
```

Any valid variable name will do, but it must have the correct number of characters. Next choose the amount by which you want succeeding line numbers to increase. For example, to renumber in increments of 1Ø, type

```
STEP = 1Ø
```

Finally, type the this command:

```
CALL -1Ø531
```

As each line of the program is renumbered, its old line number is displayed with an "arrow" pointing to the new line number. A possible example might appear like this on the APPLE's screen:

```
7->1ØØØ
213->1Ø1Ø
527->1Ø2Ø
698->1Ø3Ø
13ØØØ->1Ø4Ø
13233->1Ø5Ø
```

RENUMBERING PORTIONS OF A PROGRAM

You do not have to renumber your entire program. You can renumber just the lines numbered from, say, 3ØØ to 5ØØ by assigning values to four variables. Again, you must first type the command

```
CLR
```

to clear the BASIC variable table.

The first two variables for partial renumbering are the same as those for renumbering the whole program. They specify that the program portion, after renumbering, will begin with line number 2ØØ, say, and that each line's number thereafter will be 2Ø greater than the previous line's:

```
START = 2ØØ
STEP = 2Ø
```

The next two variables specify the program portion's range of line numbers before renumbering:

```
FROM = 3ØØ
TO = 5ØØ
```

The final command is also different. For renumbering a portion of a program, use the command:

```
CALL -1Ø521
```

If the program was previously numbered

```
 1ØØ
 12Ø
 3ØØ
 31Ø
 4Ø2
 5ØØ
2ØØØ
2Ø22
```

then after the renumbering specified above, the APPLE will show this list of changes:

```
3ØØ->2ØØ
31Ø->22Ø
4Ø2->24Ø
5ØØ->26Ø
```

and the new program line numbers will be

```
 1ØØ
 12Ø
 2ØØ
 22Ø
 24Ø
 26Ø
2ØØØ
2Ø22
```

You cannot renumber in such a way that the renumbered lines would replace, be inserted between or be intermixed with un-renumbered lines. Thus, you cannot change the order of the program lines. If you try, the message

*** RANGE ERR

is displayed after the list of proposed line changes, and the line numbers themselves are left unchanged. If you type the commands in the wrong order, nothing happens, usually.

COMMENTS:

1. If you do not CLR before renumbering, unexpected line numbers may result. It may or may not be possible to renumber the program again and save your work.

2. If you omit the START or STEP values, the computer will choose them unpredictably. This may result in loss of the program.

3. If an arithmetic expression or variable is used in a GOTO or GOSUB, that GOTO or GOSUB will generally not be renumbered correctly. For example, GOTO TEST or GOSUB 1Ø+2Ø will not be renumbered correctly.

4. Nonsense values for STEP, such as Ø or a negative number, can render your program unusable. A negative START value can renumber your program with line numbers above 32767, for what it's worth. Such line numbers are difficult to deal with. For example, an attempt to LIST one of them will result in a >32767 error. Line numbers greater than 32767 can be corrected by renumbering the entire program to lower line numbers.

5. The display of line number changes can appear correct even though the line numbers themselves have not been changed correctly. After the *** RANGE ERR message, for instance, the line numbers are left with their original numbering. LIST your program and check it before using it.

6. The Renumber feature applies only to Integer BASIC programs.

7. Occasionally, what seems to be a "reasonable" renumbering does not work. Try the renumbering again, with a different START and STEP value.

CHAPTER 2
APPEND

APPENDING ONE BASIC PROGRAM TO ANOTHER

If you have one program or program portion stored in your APPLE's memory, and another saved on tape, it is possible to combine them into one program. This feature is especially useful when a subroutine has been developed for one program, and you wish to use it in another program without retyping the subroutine.

For the Append feature to function correctly, all the line numbers of the program in memory must be greater than all the line numbers of the program to be appended from tape. In this discussion, we will call the program saved on tape "Program1," and the program in APPLE's memory "Program2."

If Program2 is not in APPLE's memory already, use the usual command

```
LOAD
```

to put Program2 (with high line numbers) into the APPLE. Using the Renumber feature, if necessary, make sure that all the line numbers in Program2 are greater than the highest line number in Program1.

Now place the tape for Program1 in the tape recorder. Use the usual loading procedure, except that instead of the LOAD command use this command:

```
CALL -11Ø76
```

This will give the normal beeps, and when the second beep has sounded, the two programs will both be in memory. If this step causes the message

```
*** MEM FULL ERR
```

to appear, neither Program2 nor Program1 will be accessible. In this case, use the command

```
CALL -11Ø59
```

to recover Program2, the program which was already in APPLE's memory.

COMMENTS:

1. The Append feature operates only with APPLE II Integer BASIC programs.

2. If the line numbers of the two progams are not as described, expect unpredictable results.

CHAPTER 3
TAPE VERIFY (BASIC)

VERIFYING A BASIC PROGRAM SAVED ON TAPE

Normally, it is impossible (unless you have two APPLEs) to know whether or not you have successfully saved your current program on tape, in time to do something about a defective recording. The reason is this: when you SAVE a program on tape, the only way to discover whether it has been recorded correctly is to LOAD it back in to the APPLE. But, when you LOAD a program, the first thing the APPLE does is erase whatever current program is stored. So, if the tape is bad, you only find out after your current program has been lost.

The Tape Verify feature solves this problem. Save your current program in the usual way:

```
SAVE
```

Rewind the tape, and (without modifying your current program in any way) type the command

```
CALL -1Ø955
```

Do not press the RETURN key until after you start the tape playing. If the tape reads in normally (with the usual two beeps), then it is correct. If there is any error on the tape, you will get a beep and the ERR message. If this happens, you will probably want to try re-recording the tape, although you don't know for sure whether the Tape Verify error means that the tape wasn't recorded right or if it just didn't play back properly. In any case, if it does verify, you know that it is good.

COMMENTS:

1. This works only with Integer BASIC programs.

2. Any change in the program, however slight, between the time the program is SAVEd on tape and the time the tape is verified, will cause the verification to fail.

CHAPTER 4
TAPE VERIFY (Machine Code or Data)

VERIFYING A PORTION OF MEMORY SAVED ON TAPE

Users of machine-language routines will find that this version of the Tape Verify feature meets their needs. Save the desired portion of memory, from address1 to address2, in the usual way:

address1 . address2 W return

Note: the example instructions in this chapter often include spaces for easier reading; do not type these spaces.

Rewind the tape, and type (after the asterisk prompt)

D52EG return

This initializes the Tape Verify feature by preparing locations $3F8 through $3FA for the ctrl Y vector. Now type (do not type the spaces)

address1 . address2 ctrl Y return

and re-play the tape. The first error encountered stops the program and is reported with a beep and the word ERR. If it is not a checksum error, then the Tape Verify feature will print out the location where the tape and memory disagreed and the data that it expected on the tape.

Note: type "ctrl Y" by typing Y while holding down the CTRL key; ctrl Y is not displayed on the TV screen. Type "return" by pressing the RETURN key.

COMMENTS:

Any change in the specified memory area, however slight, between the time the program is saved on tape and the time the tape is verified, will cause the verification to fail.

CHAPTER 5
RELOCATE

PART A: THEORY OF OPERATION

RELOCATING MACHINE-LANGUAGE CODE

Quite frequently, programmers encounter situations that call for relocating machine-language (not BASIC) programs on the 65Ø2-based APPLE II computer. Relocation implies creating a new version of the program, a version that runs properly in an area of memory different from that in which the original program ran.

If they rely on the relative branch instruction, certain small 65Ø2 programs can simply be moved without alteration, using the existing Monitor Move commands. Other programs will require only minor hand-modification after Monitor Moving. These modifications are simplified on the APPLE II by the built-in disassembler, which pinpoints absolute memory-reference instructions such as JMP's and JSR's.

However, sometimes it is necessary to relocate lengthy programs containing multiple data segments interspersed with code. Using this Machine-Code Relocation feature can save you hours of work on such a move, with improved reliability and accuracy.

The following situations call for program relocation:

1. Two different programs, which were originally written to run in identical memory locations, must now reside and run in memory concurrently.

2. A program currently runs from ROM. In order to modify its operation experimentally, a version must be generated which runs from a different set of addresses in RAM.

3. A program currently running in RAM must be converted to run from EPROM or ROM addresses.

4. A program currently running on a 16K machine must be relocated in order to run on a 4K machine. Furthermore, the relocation may have to be performed on the smaller machine.

5. Because of memory-mapping differences, a program that ran on an APPLE I (or other 65Ø2-based computer) falls into unusable address space on an APPLE II.

6. Because different operating systems assign variables differently, either page-zero or non-page-zero variable allocation for a specific program may have to modified when moving the program from one make of computer to another.

7. A program, which exists as several chunks strewn about memory, must be combined in a single, contiguous block.

8. A program has outgrown the available memory space and must be relocated to a larger, "free" memory space.

9. A program insertion or deletion requires a portion of the program to move a few bytes up or down.

1Ø. On a whim, the user wishes to move a program.

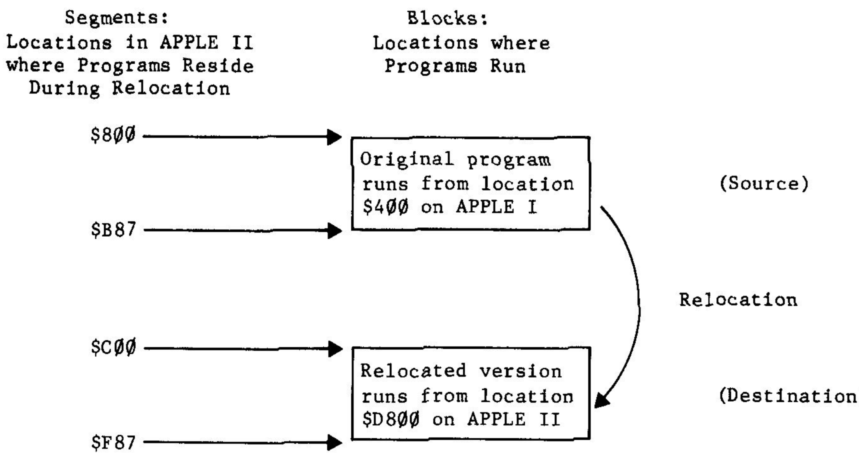
PROGRAM MODEL

Here is one simple way to visualize program relocation: starting with a program which resides and runs in a "Source Block" of memory, relocation creates a modified version of that program which resides and runs properly in a "Destination Block" of memory.

However, this model does not sufficiently describe situations where the "Source Block" and the "Destination Block" are the same locations in memory. For example, a program written to begin at location $4ØØ on an APPLE I (the $ indicates a hexadecimal number) falls in the APPLE II screen-memory range. It must be loaded to some other area of memory in the APPLE II. But the program will not run properly in its new memory locations, because various absolute memory references, etc., are now wrong. This program can then be "relocated" right back into the same new memory locations, a process which modifies it to run properly in its new location.

A more versatile program model is as follows. A program or section of a program written to run in a memory range termed the "Source Block" actually resides currently in a range termed the "Source Segments". Thus a program written to run from location $4ØØ may currently reside beginning at location $8ØØ. After relocation, the new version of the program must be written to run correctly in a range termed the "Destination Block" although it will actually reside currently in a range termed the "Destination Segments". Thus a program may be relocated such that it will run correctly from location $D8ØØ (a ROM address) yet reside beginning at location $CØØ prior to being saved on tape or used to burn EPROMs (obviously, the relocated program cannot immediately reside at locations reserved for ROM). In some cases, the Source and Destination Segments may overlap.

BLOCKS AND SEGMENTS EXAMPLE



SOURCE BLOCK: $4ØØ-$787

DESTINATION BLOCK: $D8ØØ-$DB87

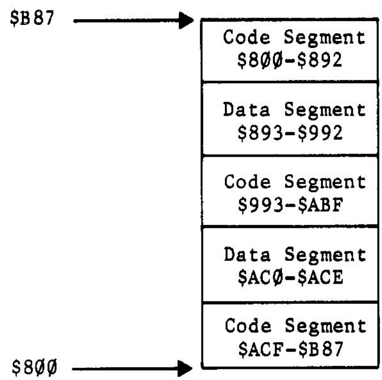
SOURCE SEGMENTS: $8ØØ-$B87

DESTINATION SEGMENTS: $CØØ-$F87

DATA SEGMENTS

The problem with relocating a large program all at once is that blocks of data (tables, text, etc.) may be interspersed throughout the code. During relocation, this data may be treated as if it were code, causing the data to be changed or causing code to be altered incorrectly because of boundary uncertainties introduced when the data takes on the multi-byte attribute of code. This problem is circumvented by dividing the program into code segments and data segments, and then treating the two types of segment differently.

CODE AND DATA SEGMENTS EXAMPLE



The Source Code Segments are relocated (using the 65Ø2 Code-Relocation feature), while the Source Data Segments are moved (using the Monitor Move command).

HOW TO USE THE CODE-RELOCATION FEATURE

1. To initialize the 6502 Code-Relocation feature, press the RESET key to invoke the Monitor, and then type

D4D5G return

The Monitor user function ctrl Y will now call the Code-Relocation feature as a subroutine at location $3F8.

Note: To type "ctrl Y", type Y while holding down the CTRL key. To type "return", press the RETURN key. In the remainder of this discussion, all instructions are typed to the right of the Monitor prompt character (*). The example instructions in this chapter often include spaces for easier reading; do not type these spaces.

2. Load the source program into the "Source Segments" area of memory (if it is not already there). Note that this need not be where the program normally runs.

3. Specify the Destination and Source Block parameters. Remember that a Block refers to locations from which the program will run, not the locations at which the Source and Destination Segments actually reside during the relocation. If only a portion of a program is to be relocated, then that portion alone is specified as the Block.

DEST BLOCK BEG < SOURCE BLOCK BEG . SOURCE BLOCK END ctrl Y * return

Notes: the syntax of this command closely resembles that of the Monitor Move command. Type "ctrl Y" by pressing the Y key while holding down the CTRL key. Then type an asterisk (*); and finally, type "return" by pressing the RETURN key. Do not type any spaces within the command.

4. Move all Data Segments and relocate all Code Segments in sequential (increasing address) order. It is wise to prepare a list of segments, specifying beginning and ending addresses, and whether each segment is code or data.

If First Segment is Code:

DEST SEGMENT BEG < SOURCE SEGMENT BEG . SOURCE SEGMENT END ctrl Y return

If First Segment is Data:

DEST SEGMENT BEG < SOURCE SEGMENT BEG . SOURCE SEGMENT END M return

After the first segment has been either relocated (if Code) or Moved (if data), subsequent segments can be relocated or Moved using a shortened form of the command.

Subsequent Code Segments:

. SOURCE SEGMENT END ctrl Y return (Relocation)

Subsequent Data Segments:

. SOURCE SEGMENT END M return (Move)

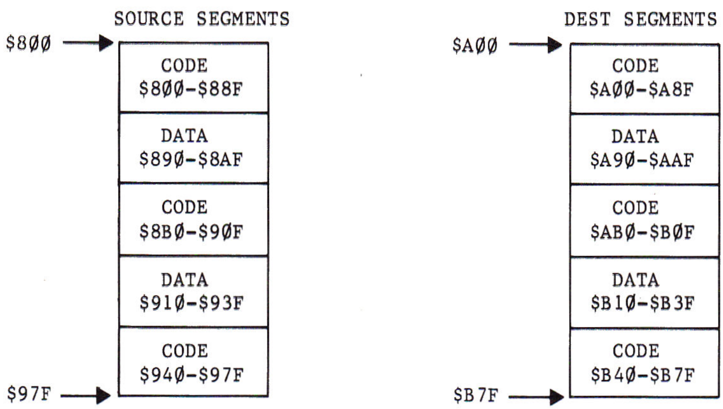
Note: the shortened form of the command can only be used if each "subsequent" segment is contiguous to the segment previously relocated or Moved. If a "subsequent" segment is in a part of memory that does not begin exactly where the previous segment ended, it must be Moved or relocated using the full "First Segment" format.

If the relocation is performed "in place" (SOURCE and DEST SEGMENTs reside in identical locations) then the SOURCE SEGMENT BEG parameter may be omitted from the First Segment relocate or Move command.

PART B: CODE-RELOCATION EXAMPLES

EXAMPLE 1. Straightforward Relocation

Program A resides and runs in locations $8ØØ-$97F. The relocated version will reside and run in locations $AØØ-$B7F.



SOURCE BLOCK:	$8ØØ-$97F	DEST BLOCK:	$AØØ-$B7F
SOURCE SEGMENTS:	$8ØØ-$97F	DEST SEGMENTS:	$AØØ-$B7F

(a) Initialize Code-Relocation feature:

```
reset D4D5G return
```

(b) Specify Destination and Source Block parameters (locations from which the program will run):

```
AØØ < 8ØØ . 97F  ctrl Y  *  return
```

(c) Relocate first segment (code):

```
AØØ < 8ØØ . 88F  ctrl Y  return
```

(d) Move subsequent Data Segments and relocate subsequent Code Segments, in ascending address sequence:

```
. 8AF  M  return                        (data)
. 9ØF  ctrl Y  return                   (code)
. 93F  M  return                        (data)
. 97F  ctrl Y  return                   (code)
```

Note that step (d) illustrates abbreviated versions of the following commands:

```
A9Ø < 89Ø . 8AF  M  return              (data)
ABØ < 8BØ . 9ØF  ctrl Y  return         (code)
B1Ø < 91Ø . 93F  M  return              (data)
B4Ø < 94Ø . 97F  ctrl Y  return         (code)
```

EXAMPLE 2. Index into Block

Suppose that the program of Example 1 uses an indexed reference into the Data Segment at $89Ø as follows:

```
LDA  7BØ,X
```

where the X-REG is presumed to contain a number in the range $EØ to $FF. Because address $7BØ is outside the Source Block, it will not be relocated. This may be handled in one of two ways.

(a) You may fix the exception by hand; or

(b) You may begin the Block specifications one page lower than the addresses at which the original and relocated programs begin to use all such "early references." One lower page is enough, since FF (the number of bytes in one page) is the largest offset number that the X-REG can contain. In EXAMPLE 1, change step (b) to:

```
9ØØ < 7ØØ . 97F  ctrl Y  *  return
```

Note: with this Block specification, <u>all</u> program references to the "prior page" (in this case the $7ØØ page) will be relocated.

EXAMPLE 3. Immediate Address References

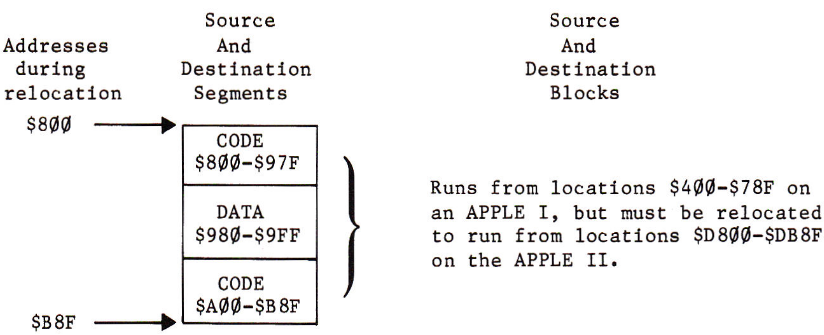
Suppose that the program of EXAMPLE 1 has an immediate reference which is an address. For example,

```
LDA   #$3F
STA   LOCØ
LDA   #$Ø8
STA   LOC1
JMP   (LOCØ)
```

In this example, the LDA #$Ø8 will not be changed during relocation and the user will have to hand-modify it to $ØA.

EXAMPLE 4. Unusable Block Ranges

Suppose a program was written to run from locations $4ØØ-$78F on an APPLE I. A version which will run in ROM locations $D8ØØ-$DB8F must be generated. The Source (and Destination) Segments will reside in locations $8ØØ-$B8F on the APPLE II during relocation.



SOURCE BLOCK: $4ØØ-$78F
SOURCE SEGMENTS: $8ØØ-$B8F

DEST BLOCK: $D8ØØ-$DB8F
DEST SEGMENTS: $8ØØ-$B8F

(a) Initialize the Code-Relocation feature:

reset D4D5G return

(b) Load original program into locations $8ØØ-$B8F (despite the fact that it doesn't run there):

8ØØ . B8F R return

(c) Specify Destination and Source Block parameters (locations from which the original and relocated versions will run):

```
D8ØØ < 4ØØ . 78F  ctrl Y     return
```

(d) Move Data Segments and relocate Code Segments, in ascending address sequence:

```
8ØØ < 8ØØ . 97F  ctrl Y  return              (first segment, code)
. 9FF  M  return                             (data)
. B8F  ctrl Y  return                        (code)
```

Note that because the relocation is done "in place", the SOURCE SEGMENT BEG parameter is the same as the DEST SEGMENT BEG parameter ($8ØØ) and need not be specified. The initial segment relocation command may be abbreviated as follows:

```
8ØØ < . 97F  ctrl Y  return
```

EXAMPLE 5. Changing the Page Zero Variable Allocation

Suppose the program of EXAMPLE 1 need not be relocated, but the page zero variable allocation is from $2Ø to $3F. Because these locations are reserved for the APPLE II system monitor, the allocation must be changed to locations $8Ø-$9F. The Source and Destination Blocks are thus not the program but rather the variable area.

SOURCE BLOCK:	$2Ø-$3F	DEST BLOCK:	$8Ø-$9F
SOURCE SEGMENTS:	$8ØØ-$97F	DEST SEGMENTS:	$8ØØ-$97F

(a) Initialize the Code-Relocation feature:

```
reset D4D5G return
```

(b) Specify Destination and Source Blocks:

```
8Ø < 2Ø . 3F  ctrl Y  *  return
```

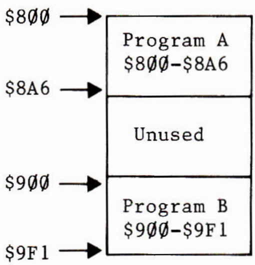
(c) Relocate Code Segments and Move Data Segments, in place:

```
8ØØ < . 88F  ctrl Y  return            (first segment, code)
. 8AF  M  return                       (data)
. 9ØF  ctrl Y  return                  (code)
. 93F  M  return                       (data)
. 97F  ctrl Y  return                  (code)
```

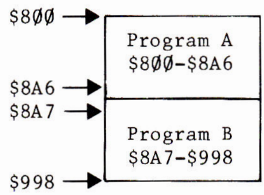
EXAMPLE 6. Split Blocks with Cross-Referencing

Program A resides and runs in locations $8ØØ-$8A6. Program B resides and runs in locations $9ØØ-$9F1. A single, contiguous program is to be generated by moving Program B so that it immediately follows Program A. Each of the programs contains references to memory locations within the other. It is assumed that the programs contain no Data Segments.

SOURCE SEGMENTS



DEST SEGMENTS



SOURCE BLOCK:	$9ØØ-$9F1	DEST BLOCK:	$8A7-$998
SOURCE SEGMENTS:	$8ØØ-$8A6 (A)	DEST SEGMENTS:	$8ØØ-$8A6 (A)
	$9ØØ-$9F1 (B)		$8A7-$998 (B)

(a) Initialize the Code-Relocation feature:

D4D5G return

(b) Specify Destination and Source Blocks (Program B only):

8A7 < 9ØØ . 9F1 ctrl Y * return

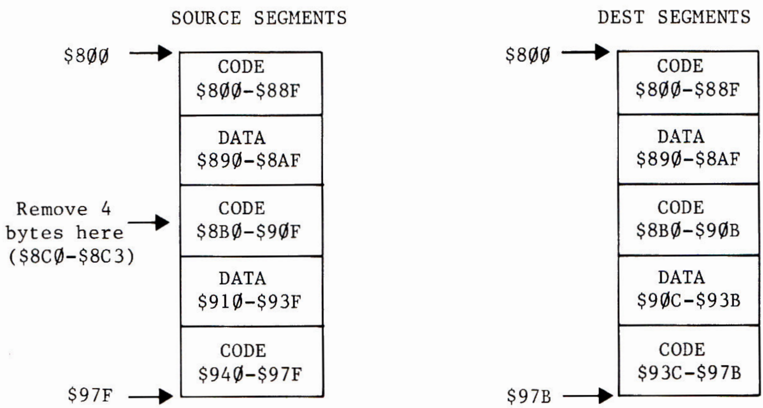
(c) Relocate each of the two programs individually. Program A must be relocated even though it does not move.

8ØØ < . 8A6 ctrl Y return (program A, "in place")
8A7 < 9ØØ . 9F1 ctrl Y return (program B, not "in place")

Note that any Data Segments within the two programs would necessitate additional relocation and Move commands.

EXAMPLE 7. Code Deletion

Four bytes of code are to be removed from within a program, and the program is to contract accordingly.




```
   SOURCE BLOCK:   $8C4-$97F                DEST BLOCK:   $8CØ-$97B
SOURCE SEGMENTS:   $8ØØ-$88F (code)     DEST SEGMENTS:   $8ØØ-$88F (code)
                   $89Ø-$8AF (data)                      $89Ø-$8AF (data)
                   $8BØ-$8BF (code)                      $8BØ-$8BF (code)
                   $8C4-$9ØF (code)                      $8CØ-$9ØB (code)
                   $91Ø-$93F (data)                      $9ØC-$93B (data)
                   $94Ø-$97F (code)                      $93C-$97B (code)
```

(a) Initialize Code-Relocation feature:

```
reset D4D5G return
```

(b) Specify Destination and Source Blocks:

```
8CØ < 8C4 . 97F  ctrl Y  *  return
```

(c) Relocate Code Segments and Move Data Segments, in ascending address sequence:

```
8ØØ <. 88F  ctrl Y  return          (first segment, code, "in place")
. 8AF  M  return                        (data)
. 8BF  ctrl Y  return                   (code)
8CØ < 8C4 . 9ØF  ctrl Y  return     (first segment, code, not "in place")
. 93F  M  return                        (data)
. 97F  ctrl Y  return                   (code)
```

(d) Relative branches crossing the deletion boundary will be incorrect, since the relocation process does not modify them (only zero-page and absolute memory references). The user must patch these by hand.

EXAMPLE 8. Relocating the APPLE II Monitor ($F800–$FFFF) to Run in RAM ($800–$FFF)

```
   SOURCE BLOCK:  $F7ØØ-$FFFF              DEST BLOCK:  $7ØØ-$FFF
                           (see EXAMPLE 2)

SOURCE SEGMENTS:  $F8ØØ-$F961 (code)   DEST SEGMENTS:  $8ØØ-$961 (code)
                  $F962-$FA42 (data)                   $962-$A42 (data)
                  $FA43-$FB18 (code)                   $A43-$B18 (code)
                  $FB19-$FB1D (data)                   $B19-$B1D (data)
                  $FB1E-$FFCB (code)                   $B1E-$FCB (code)
                  $FFCC-$FFFF (data)                   $FCC-$FFF (data)

IMMEDIATE ADDRESS REFERENCES (see EXAMPLE 3):     $FFBF
                                                  $FEA8
                                         (more if not relocating
                                            to page boundary)
```

(a) Initialize the Code-Relocation feature:

```
reset D4D5G return
```

(b) Specify Destination and Source Block parameters:

```
7ØØ < F7ØØ . FFFF  ctrl Y  *  return
```

(c) Relocate Code Segments and move Data Segments, in ascending address sequence:

```
8ØØ < F8ØØ . F961  ctrl Y  return          (first segment, code)
. FA42  M  return                          (data)
. FB18  ctrl Y  return                     (code)
. FB1D  M  return                          (data)
. FFCB  ctrl Y  return                     (code)
. FFFF  M  return                          (data)
```

(d) Change immediate address references:

```
FBF : E  return          (was $FE)
EA8 : E  return          (was $FE)
```

PART C: PLOTTING POINTS AND LINES

TECHNICAL INFORMATION

The following details illustrate special technical features of the APPLE II which are used by the Code-Relocation feature.

1. The APPLE II Monitor command

Addr4 < Addr1 . Addr2 ctrl Y return (Addr1, Addr2, and Addr4 are addresses)

vectors to location $3F8 with the value Addr1 in locations $3C (low) and $3D (high), Addr2 in locations $3E (low) and $3F (high), and Addr4 in locations $42 (low) and $43 (high). Location $34 (YSAV) holds an index to the next character of the command buffer (after the ctrl Y). The command buffer (IN) begins at $2ØØ.

2. If ctrl Y is followed by * , then the Block parameters are simply preserved as follows:

Parameter	Preserved at	SWEET16 Reg Name
DEST BLOCK BEG	$8, $9	TOBEG
SOURCE BLOCK BEG	$2, $3	FRMBEG
SOURCE BLOCK END	$4, $5	FRMEND

3. If ctrl Y is not followed by * , then a segment relocation is initiated at RELOC2 ($3BB). Throughout, Addr1 ($3C, $3D) is the Source Segment pointer and Addr4 ($42, $43) is the Destination Segment pointer.

4. INSDS2 is an APPLE II Monitor subroutine which determines the length of a 65Ø2 instruction, given the opcode in the A-REG, and stores that opcode's instruction length in the variable LENGTH (location $2F) .

Instruction Type in A-REG	LENGTH (in $2F)
Invalid	Ø
1 byte	Ø
2 byte	1
3 byte	2

5. The code from XLATE to SW16RT ($3D9-$3E6) uses the APPLE II 16-bit interpretive machine, SWEET16. The target address of the 65Ø2 instruction being relocated (locations $C low and $D high) occupies the SWEET16 register named ADR. If ADR is between FRMBEG and FRMEND (inclusive) then it is replaced by

ADR - FRMBEG + TOBEG

6. NXTA4 is an APPLE II Monitor subroutine which increments Addr1 (Source Segment index) and Addr4 (Destination Segment index). If Addr1 exceeds Addr2 (Source Segment end), then the carry is set; otherwise, it is cleared.

ALGORITHM USED BY THE CODE-RELOCATION FEATURE

1. Set SOURCE PTR to beginning of Source Segment and DEST PTR to beginning of Destination Segment.

2. Copy 3 bytes from Source Segment (using SOURCE PTR) to temp INST area.

3. Determine instruction length from opcode (1, 2 or 3 bytes).

4. If two-byte instruction with non-zero-page addressing mode (immediate or relative) then go to step 7.

5. If two-byte instruction then clear 3rd byte so address field is Ø-255 (zero page).

6. If address field (2nd and 3rd bytes of INST area) falls within Source Block, then substitute

   ADR - SOURCE BLOCK BEG + DEST BLOCK BEG

7. Move "length" bytes from INST area to Destination Segment (using DEST PTR). Update SOURCE and DEST PTR's by length.

8. If SOURCE PTR is less than or equal to SOURCE SEGMENT END then goto step 2., else done.

COMMENTS:

Each Move or relocation is carried out sequentially, one byte at a time, beginning with the byte at the smallest source address. As each source byte is Moved or relocated, it overwrites any information that was in the destination location. This is usually acceptable in these kinds of Moves and relocations:

1. Source Segments and Destination Segments do not share any common locations (no source location is overwritten).

2. Source Segments are in locations identical to the locations of the Destination Segments (each source byte overwrites itself).

3. Source Segments are in locations whose addresses are larger than the addresses of the Destination Segments' locations (any overwritten source bytes have already been Moved or relocated). This is a move toward smaller addresses.

If, however, the Source Segments and the Destination Segments share some common locations, and the Source Segments occupy locations whose addresses are smaller than the addresses of the Destination Segments' locations, then the source bytes occupying the common locations will be overwritten before they are Moved or relocated. If you attempt such a relocation, you will lose your program and data in the memory area common to both Source Segments and Destination Segments. To accomplish a small Move or relocation toward larger addresses, you must Move or relocate to an area of memory well away from the Source Segments (no address in common); then Move the entire relocated program back to its final resting place.

Note: the example instructions in this chapter often include spaces for easier reading; do not type these spaces.

CHAPTER 6
RAM TEST

TESTING THE APPLE'S MEMORY

With this program, you can easily discover any problems in the RAM (for Random Access Memory) chips in your APPLE. This is especially useful when adding new memory. While a failure is a rare occurrence, memory chips are both quite complex and relatively expensive. This program will point out the exact memory chip or chips, if any, that have malfunctioned.

Memory chips are made in two types: one type can store 4K (4Ø96) bits of information, the other can store 16K (16384) bits of information. Odd as it seems, the two types look alike, except for a code number printed on them.

The APPLE has provisions for inserting as many as 24 memory chips of either type into its main printed-circuit board, in three rows of eight sockets each. An eight-bit byte of information consists of one bit taken from each of the eight memory chips in a given row. For this reason, memory can be added only in units of eight identical memory chips at a time, filling an entire row. Eight 4K memory chips together in one row can store 4K bytes of information. Eight 16K memory chips in one row can store 16K bytes of information.

Inside the APPLE II, the three rows of sockets for memory chips are row "C", row "D" and row "E". The rows are lettered along the left edge of the printed-circuit board, as viewed from the front of the APPLE. The memory chips are installed in the third through the tenth sockets (counting from the left) of rows C, D and E. These sockets are labelled "RAM". Row C must be filled; and row E may be filled only if row D is filled. Depending on the configuration of your APPLE's memory, the eight RAM sockets in a given row of memory must be filled entirely with 4K memory chips, entirely with 16K memory chips, or all eight RAM sockets may be empty.

To test the memory chips in your computer, you must first initialize the RAM Test program. Press the RESET key to invoke the Monitor, and then type

D5BCG return

Next, specify the hexadecimal starting address for the portion of memory that you wish to test. You must also specify the hexadecimal number of "pages" of memory that you wish tested, beginning at the given starting address. A page of memory is 256 bytes ($1ØØ Hex). Representing the address by "a" and the number of pages by "p" (both in hexadecimal), start the RAM test by typing

a . p ctrl Y return

Note 1: to type "ctrl Y", type Y while holding down the CTRL key; ctrl Y is not displayed on the TV screen. Type "return" by pressing the RETURN key. The example instructions in this chapter often include spaces for easier reading; do not type these spaces.

Note 2: test length p*1ØØ must not be greater than starting address a.

For example,

2ØØØ.1Ø ctrl Y return

tests hexadecimal 1ØØØ bytes of memory (4Ø96, or "4K" bytes, in decimal), starting at hexadecimal address 2ØØØ (8192, or "8K", in decimal).

If the asterisk returns (after a delay that may be a half minute or so) without an error message (see ERROR MESSAGES discussion), then the specified portion of memory has tested successfully.

TABLE OF ADDRESS RANGES FOR STANDARD RAM CONFIGURATIONS

If the 3 Memory Configuration Blocks Look like this:	Then Row of Memory	Contains this Range of Hexadecimal RAM Addresses	And the total System Memory, If this is last Row filled, is
4K	C	ØØØØ-ØFFF	4K
4K	D	1ØØØ-1FFF	8K
4K	E	2ØØØ-2FFF	12K
16K	C	ØØØØ-3FFF	16K
4K	D	4ØØØ-4FFF	2ØK
4K	E	5ØØØ-5FFF	24K
16K	C	ØØØØ-3FFF	16K
16K	D	4ØØØ-7FFF	32K
16K	E	8ØØØ-BFFF	48K

A 4K RAM Row contains 1Ø Hex pages (hex 1ØØØ bytes, or decimal 4Ø96 bytes). A 16K RAM Row contains 4Ø Hex pages (hex 4ØØØ bytes, or decimal 16384 bytes).

A complete test for a 48K system would be as follows:

```
 4ØØ.4   ctrl Y return  <---- This tests the screen area of memory
 8ØØ.8   ctrl Y return  }     These first four tests examine
1ØØØ.1Ø  ctrl Y return  } <----          the first 16K row of memory   (Row C)
2ØØØ.2Ø  ctrl Y return  }
4ØØØ.4Ø  ctrl Y return  <---- This tests the second 16K row of memory (Row D)
8ØØØ.4Ø  ctrl Y return  <---- This tests the third 16K row of memory  (Row E)
```

Systems containing more than 16K of memory should also receive the following special test that looks for problems at the boundary between rows of memory:

3ØØØ.2Ø ctrl Y return

Systems containing more than 32K of memory should receive the previous special test, plus the following:

7ØØØ.2Ø ctrl Y return

Tests may be run separately or they may be combined into one instruction. For instance, for a 48K system you can type:

4ØØ.4 ctrl Y 8ØØ.8 ctrl Y 1ØØØ.1Ø ctrl Y 2ØØØ.2Ø ctrl Y 3ØØØ.2Ø ctrl Y 4ØØØ.4Ø ctrl Y 7ØØØ.2Ø ctrl Y 8ØØØ.4Ø ctrl Y return

Remember, ctrl Y will not print on the screen, but it <u>must</u> be typed. With the single exception noted in the section TESTING FOR INTERMITTENT FAILURE, spaces are shown for easier reading but should <u>not</u> be typed.

During a full test such as the one shown above, the computer will beep at the completion of each sub-test (each sub-test ends with a ctrl Y). At the end of the full test, if no errors have been found the APPLE will beep and the blinking cursor will return with the Monitor prompt character (*). It takes approximately 5Ø seconds for the computer to test the RAM memory in a 16K system; larger systems will take proportionately longer.

ERROR MESSAGES

TYPE I - Simple Error

During testing, each memory address in the test range is checked by writing a particular number to it, then reading the number actually stored at that address and comparing the two.

A simple error occurs when the number written to a particular memory address differs from the number which is then read back from that <u>same</u> address. Simple errors are reported in the following format:

xxxx yy zz ERR r-c

where xxxx is the hexadecimal address at which the error was detected;
yy is the hexadecimal data written to that address;
zz is the hexadecimal data read back from that address; and
r-c is the row and column where the defective memory chip was found. Count from the left, as viewed from the front of the APPLE: the leftmost memory chip is in column 3, the rightmost is in column 1Ø.

Example:

2Ø1F ØØ 1Ø ERR D-7

TYPE II - Dynamic Error

This type of error occurs when the act of writing a number to one memory address causes the number read from a different address to change. If no simple error is detected at a tested address, all the addresses that differ from the tested address by one bit are read for changes indicating dynamic errors. Dynamic errors are reported in the following format:

xxxx yy zz vvvv qq ERR r-c

where xxxx is the hexadecimal address at which the error was detected;
yy is the hexadecimal data written earlier to address xxxx;
zz is the hexadecimal data now read back from address xxxx;
vvvv is the current hexadecimal address to which data qq was successfully written;
qq is the hexadecimal data successfully written to, and read back from, address vvvv; and
r-c is the row and column where the defective memory chip was found. Count from the left, as viewed from the front of the APPLE: the leftmost memory chip is in column 3, the rightmost is in column 1Ø. In this type of error, the indicated row (but not the column) may be incorrect.

This is similar to Type I, except that the appearance of vvvv and qq indicates an error was detected at address xxxx after data was successfully written at address vvvv.

Example:

5Ø51 ØØ Ø8 5451 ØØ ERR E-6

After a dynamic error, the indicated row (but not the column) may be incorrect. Determine exactly which tests check each row of chips (according to the range of memory addresses corresponding to each row), and run those tests by themselves. Confirm your diagnosis by replacing the suspected memory chip with a known good memory chip (you can use either a 4K or a 16K memory chip, for this replacement). Remember to turn off the APPLE's power switch and to discharge yourself before handling the memory chips.

TESTING FOR INTERMITTENT FAILURE (Automatically Repeating Test)

This provides a way to test memory over and over again, indefinitely. You will type a complete series of tests, just as you did before, except that you will:

a. precede the complete test with the letter N
b. follow the complete test with 34:Ø
c. type at least one space before pressing the RETURN key.

Here is the format:

N (memory test to be repeated) 34:Ø (type one space) return

NOTE: You must type at least one space at the end of the line, prior to pressing the RETURN key. This is the only space that should be typed (all other spaces shown within instructions in this chapter are for easier reading only; they should not be typed).

Example (for a 48K system):

N 4ØØ.4 ctrl Y 8ØØ.8 ctrl Y 1ØØØ.1Ø ctrl Y 2ØØØ.2Ø ctrl Y 3ØØØ.2Ø ctrl Y 4ØØØ.4Ø ctrl Y 7ØØØ.2Ø ctrl Y 8ØØØ.4Ø ctrl Y 34:Ø   return

Run this test for at least one hour (preferably overnight) with the APPLE's lid in place. This allows the system and the memory chips to reach maximum operating temperature.

Only if a failure occurs will the APPLE display an error message and rapidly beep three times; otherwise, the APPLE will beep once at the successful end of each sub-test. To stop this repeating test, you must press the RESET key.

COMMENTS:

1. You cannot test the APPLE's memory below the address of 4ØØ (Hex), since various pointers and other system necessities are there. In any case, if that region of memory has problems, the APPLE won't function.

2. For any subtest, the number of pages tested cannot be greater than the starting address divided by 1ØØ Hex. 2ØØØ.3Ø ctrl Y will not work, but 5ØØØ.3Ø ctrl Y will.

3. Before changing anything inside the APPLE, make sure the APPLE is plugged into a grounded, 3-wire power outlet, and that the power switch on the back of the computer is turned off. Always touch the outside metal bottom plate of the APPLE II, prior to handling any memory chips. This is done to remove any static charge that you may have acquired.

EVEN A SMALL STATIC CHARGE CAN DESTROY MEMORY CHIPS

4. Besides the eight memory chips, some additions of memory require changing three other chip-like devices called Memory Configuration Blocks. The Memory Configuration Blocks tell the APPLE which type of memory chip (4K or 16K) is to be plugged into each row of memory. A complete package for adding memory to your computer, containing all necessary parts and detailed instructions, can be purchased from APPLE Computer Inc. To add 4K of memory, order the 4K Memory Expansion Module (P/N A2MØØ14). To add 16K of memory, order the 16K Memory Expansion Module (P/N A2MØØ16).

CHAPTER 7
MUSIC

GENERATING MUSICAL TONES

The Music feature is most easily used from within an Integer BASIC program. It greatly simplifies the task of making the APPLE II into a music-playing device.

There are three things the computer needs to know before playing a note: pitch (how high or low a note), duration (how long a time it is to sound), and timbre. Timbre is the quality of a sound that allows you to distinguish one instrument from another even if they are playing at the same pitch and loudness. This Music feature does not permit control of loudness.

It is convenient to set up a few constants early in the program:

```
MUSIC = -1Ø473
PITCH = 767
TIME = 766
TIMBRE = 765
```

There are 5Ø notes available, numbered from 1 to 5Ø. The statement

```
POKE PITCH, 32
```

will set up the Music feature to produce (approximately) the note middle C. Increasing the pitch value by one increases the pitch by a semitone. Thus

```
POKE PITCH, 33
```

would set up the Music feature to produce the note C sharp. Just over four chromatic octaves are available. The note number Ø indicates a rest (a silence) rather than a pitch.

The duration of the note is set by

```
POKE TIME, t
```

Where t is a number from 1 to 255. The higher the number, the longer the note. A choice of t = 17Ø gives notes that are approximately one second long. To get notes at a metronome marking of MM, use a duration of 1Ø2ØØ/MM. For example, to get 2Ø4 notes per minute (approximately) use the command

```
POKE TIME, 1Ø2ØØ/2Ø4
```

There are five timbres, coded by the numbers 2, 8, 16, 32 and 64. They are not very different from one another. With certain timbres, a few of the extremely low or high notes do not give the correct pitch. Timbre 32 does not have this problem.

```
POKE TIMBRE, 32
```

When the pitch, time, and timbre have been set, the statement

```
CALL MUSIC
```

will cause the specified note to sound.

The following program plays a chromatic scale of four octaves:

```
10 MUSIC = -10473: PITCH = 767: TIME = 766: TIMBRE = 765
20 POKE TIME, 40: POKE TIMBRE, 32
30 FOR I = 1 TO 49
40 POKE PITCH, I
50 CALL MUSIC
60 NEXT I: END
```

Where X is a number from 51 through 255,

```
POKE PITCH, X
```

will specify various notes, in odd sequences. In the program above, change line 40 to

```
40 POKE PITCH, 86
```

for a demonstration.

COMMENTS:

Some extremely high or low notes will come out at the wrong pitch with certain timbres.

CHAPTER 8
HIGH-RESOLUTION GRAPHICS

PART A: SETTING UP PARAMETERS, SUBROUTINES, AND COLORS

Programmer's Aid #1 provides your APPLE with the ability to do high-resolution color graphics from Integer BASIC. You may plot dots, lines and shapes in a wide variety of detailed forms, in 6 different colors (4 colors on systems below S/N 6ØØØ), displayed from two different "pages" of memory. The standard low-resolution graphics allowed you to plot 4Ø squares across the screen by 47 squares from top to bottom of the screen. This high-resolution graphics display mode lets you plot in much smaller dots, 28Ø horizontally by 192 vertically. Because 8K bytes of memory (in locations from 8K to 16K, for Page 1) are dedicated solely to maintaining the high-resolution display, your APPLE must contain at least 16K bytes of memory. To use the Page 2 display (in locations from 16K to 24K), a system with at least 24K bytes of memory is needed. If your system is using the Disk Operating System (DOS), that occupies the top 1Ø.5K of memory: you will need a minimum 32K system for Page 1, or 36K for Page 1 and Page 2. See the MEMORY MAP on page 63 for more details.

POSITIONING THE HIGH-RESOLUTION PARAMETERS

The first statement of an Integer BASIC program intending to use the Programmer's Aid High-Resolution subroutines should be:

```
Ø  XØ = YØ = COLR = SHAPE = ROT = SCALE
```

The purpose of this statement is simply to place the six BASIC variable names used by the High-Resolution feature (with space for their values) into APPLE's "variable table" in specific, known locations. When line Ø is executed, the six High-Resolution graphics parameters will be assigned storage space at the very beginning of the variable table, in the exact order specified in line Ø. Your BASIC program then uses those parameter names to change the six parameter values in the variable table. However, the High-Resolution subroutines ignore the parameter names, and look for the parameter values in specific variable-table locations. That is why the program's first line must place the six High-Resolution graphics parameters in known variable-table locations. Different parameter names may be used, provided that they contain the same number of characters. Fixed parameter-name lengths are also necessary to insure that the parameter-value storage locations in the variable table do not change. For example, the name HI could be used in place of XØ, but X or XCOORD could not.

The parameters SHAPE, ROT, and SCALE are used only by the subroutines that draw shapes (DRAW and DRAW1, see PART E). These parameters may be omitted from programs using only the PLOT and LINE features:

```
0  X0 = Y0 = COLR
```

Omitting unnecessary parameter definitions speeds up the program during execution. However, you can omit only those unused parameters to the right of the last parameter which is used. Each parameter that is used must be in its proper place, relative to the first parameter in the definition list.

DEFINING SUBROUTINE NAMES

After the six parameters have been defined, the twelve High-Resolution subroutines should be given names, and these names should be assigned corresponding subroutine entry addresses as values. Once defined in this way, the various subroutines can be called by name each time they are used, rather than by numeric address. When subroutines are called by name, the program is easier to type, more likely to be error-free, and easier to follow and to debug.

```
5  INIT = -12288  : CLEAR = -12274 : BKGND = -11471
6  POSN = -11527  : PLOT = -11506  : LINE = -11500
7  DRAW = -11465  : DRAW1 = -11462
8  FIND = -11780 : SHLOAD = -11335
```

Any variable names of any length may be used to call these subroutines. If you want maximum speed, do not define names for subroutines that you will not use in your program.

DEFINING COLOR NAMES

Colors may also be specified by name, if a defining statement is added to the program. Note that GREEN is preceded by LET to avoid a SYNTAX ERROR, due to conflict with the GR command.

```
10  BLACK = 0 : LET GREEN = 42 : VIOLET = 85
11  WHITE = 127 : ORANGE = 170 : BLUE = 213
12  BLACK2 = 128 : WHITE2 = 255
```

Any integer from 0 through 255 may be used to specify a color, but most of the numbers not named above give rather unsatisfactory "colors". On systems below S/N 6000, 170 will appear as green and 213 will appear as violet.

Once again, unnecessary variable definitions should be omitted, as they will slow some programs. Therefore, a program should not define VIOLET = 85 unless it uses the color VIOLET.

The following example illustrates condensed initialization for a program using only the INIT, PLOT, and DRAW subroutines, and the colors GREEN and WHITE.

```
Ø  XØ = YØ = COLR = SHAPE = ROT = SCALE
5  INIT = -12288 : PLOT = -115Ø6 : DRAW = -11465
1Ø  LET GREEN = 42 : WHITE = 127

    (Body of program would go here)
```

SPEEDING UP YOUR PROGRAM

Where maximum speed of execution is necessary, any of the following techniques will help:

1. Omit the name definitions of colors and subroutines, and refer to colors and subroutines by numeric value, not by name.

2. Define the most frequently used program variable names before defining the subroutine and color names (lines 5 through 12 in the previous examples). The example below illustrates how to speed up a program that makes very frequent use of program variables I, J, and K:

```
Ø  XØ = YØ = COLR = SHAPE = ROT = SCALE
2  I = J = K
5  INIT = -12288 : CLEAR = -12274
6  BKGND = -11471 : POSN = -11527
1Ø  BLACK = Ø : VIOLET = 85
```

3. Use the High-Resolution graphics parameter names as program variables when possible. Because they are defined first, these parameters are the BASIC variables which your program can find fastest.

PART B: PREPARING THE SCREEN FOR GRAPHICS

THE INITIALIZATION SUBROUTINE

In order to use CLEAR, BKGND, POSN, PLOT, or any of the other High-Resolution subroutine CALLs, the INITialization subroutine itself must first be CALLed:

```
CALL INIT
```

The INITialization subroutine turns on the high-resolution display and clears the high-resolution screen to black. INIT also sets up certain variables necessary for using the other High-Resolution subroutines. The display consists of a graphics area that is 28Ø x-positions wide (XØ=Ø through XØ=279) by 16Ø y-positions high (YØ=Ø through YØ=159), with an area for four lines of text at the bottom of the screen. YØ values from Ø through 191 may be used, but values greater than 159 will not be displayed on the screen. The graphics origin (XØ=Ø, YØ=Ø) is at the top left corner of the screen.

CHANGING THE GRAPHICS SCREEN

If you wish to devote the entire display to graphics (28Ø x-positions wide by 192 y-positions high), use

```
POKE -163Ø2, Ø
```

The split graphics-plus-text mode may be restored at any time with

```
POKE -163Ø1, Ø
```

or another

```
CALL INIT
```

When the High-Resolution subroutines are first initialized, all graphics are done in Page 1 of memory ($2ØØØ-3FFF), and only that page of memory is displayed. If you wish to use memory Page 2 ($4ØØØ-5FFF), two POKEs allow you to do so:

```
POKE 8Ø6, 64
```

causes subsequent graphics instructions to be executed in Page 2, unless those instructions attempt to continue an instruction from Page 1 (for instance, a LINE is always drawn on the same memory page where the last previous point was plotted). After this POKE, the display will still show memory Page 1.

To see what you are plotting on Page 2,

```
POKE -16299, Ø
```

will cause Page 2 to be displayed on the screen. You can switch the screen display back to memory Page 1 at any time, with

```
POKE -163ØØ, Ø
```

while

```
POKE 8Ø6, 32
```

will return you to Page 1 plotting. This last POKE is executed automatically by INIT.

CLEARING THE SCREEN

If at any time during your program you wish to clear the current plotting page to black, use

```
CALL CLEAR
```

This immediately erases anything plotted on the current plotting page. INIT first resets the current plotting page to memory Page 1, and then clears Page 1 to black.

The entire current plotting page can be set to any solid background color with the BKGND subroutine. After you have INITialized the High-Resolution subroutines, set COLR to the background color you desire, and then

```
CALL BKGND
```

The following program turns the entire display violet:

```
Ø  XØ = YØ = COLR : REM  SET PARAMETERS
5  INIT = -12288 : BKGND = -11471 : REM  DEFINE SUBROUTINES
1Ø  VIOLET = 85 : REM  DEFINE COLOR
2Ø  CALL INIT : REM  INITIALIZE HIGH-RESOLUTION SUBROUTINES
3Ø  COLR = VIOLET : REM  ASSIGN COLOR VALUE
4Ø  CALL BKGND : REM  MAKE ALL OF DISPLAY VIOLET
5Ø  END
```

PART C: PLOTTING POINTS AND LINES

Points can be plotted anywhere on the high-resolution display, in any valid color, with the use of the PLOT subroutine. The PLOT subroutine can only be used after a CALL INIT has been executed, and after you have assigned appropriate values to the parameters XØ, YØ and COLR. XØ must in the range from Ø through 279, YØ must be in the range from Ø through 191, and COLR must be in the range from Ø through 255, or a

```
*** RANGE ERR
```

message will be displayed and the program will halt.

The program below plots a white dot at X-coordinate 35, Y-coordinate 55, and a violet dot at X-coordinate 85, Y-coordinate 9Ø:

```
Ø  XØ = YØ = COLR : REM  SET PARAMETERS
5  INIT = -12288 : PLOT = -115Ø6 : REM  DEFINE SUBROUTINES
1Ø  WHITE = 127 : VIOLET = 85 : REM  DEFINE COLORS
2Ø  CALL INIT : REM  INITIALIZE SUBROUTINES
3Ø  COLR = WHITE : REM  ASSIGN PARAMETER VALUES
4Ø  XØ = 35 : YØ = 55
5Ø  CALL PLOT : REM  PLOT WITH ASSIGNED PARAMETER VALUES
6Ø  COLR = VIOLET : REM  ASSIGN NEW PARAMETER VALUES
7Ø  XØ = 85 : YØ = 9Ø
8Ø  CALL PLOT : REM  PLOT WITH NEW PARAMETER VALUES
9Ø  END
```

The subroutine POSN is exactly like PLOT, except that nothing is placed on the screen. COLR must be specified, however, and a subsequent DRAW1 (see PART E) will take its color from the color used by POSN. This subroutine is often used when establishing the origin-point for a LINE.

Connecting any two points with a straight line is done with the LINE subroutine. As with the PLOT subroutine, a CALL INIT must be executed, and XØ, YØ, and COLR must be specified. In addition, before the LINE subroutine can be CALLed, the line's point of origin must have been plotted with a CALL PLOT or as the end point of a previous line or shape. Do not attempt to use CALL LINE without first plotting a point for the line's origin, or the line may be drawn in random memory locations, not necessarily restricted to the current memory page. Once again, XØ and YØ (the coordinates of the termination point for the line), and COLR must be assigned legitimate values, or an error may occur.

The following program draws a grid of green lines vertically and violet lines horizontally, on a white background:

```
Ø  XØ = YØ = COLR : REM  SET PARAMETERS, THEN DEFINE SUBROUTINES
5  INIT = -12288 : BKGND = -11471 : PLOT = -115Ø6 : LINE = -115ØØ
1Ø  LET GREEN = 42 : VIOLET = 85 : WHITE = 127 : REM DEFINE COLORS
2Ø  CALL INIT : REM  INITIALIZE HIGH-RESOLUTION SUBROUTINES
3Ø  POKE -163Ø2, Ø : REM  SET FULL-SCREEN GRAPHICS
4Ø  COLR = WHITE : CALL BKGND : REM  MAKE THE DISPLAY ALL WHITE
5Ø  COLR = GREEN : REM  ASSIGN PARAMETER VALUES
6Ø  FOR XØ = Ø TO 27Ø STEP 1Ø
7Ø  YØ = Ø : CALL PLOT : REM  PLOT A STARTING-POINT AT TOP OF SCREEN
8Ø  YØ = 19Ø : CALL LINE : REM  DRAW A VERTICAL LINE TO BOTTOM OF SCREEN
9Ø  NEXT XØ : REM  MOVE RIGHT AND DO IT AGAIN
1ØØ  COLR = VIOLET : REM  ASSIGN NEW PARAMETER VALUES
11Ø  FOR YØ = Ø TO 19Ø STEP 1Ø
12Ø  XØ = Ø : CALL PLOT : REM  PLOT A STARTING-POINT AT LEFT EDGE OF SCREEN
13Ø  XØ = 27Ø : CALL LINE : REM  PLOT A HORIZONTAL LINE TO RIGHT EDGE
14Ø  NEXT YØ : REM  MOVE DOWN AND DO IT AGAIN
15Ø  END
```

PART D: CREATING, SAVING AND LOADING SHAPES

INTRODUCTION

The High-Resolution feature's subroutines provide the ability to do a wide range of high-resolution graphics "shape" drawing. A "shape" is considered to be any figure or drawing (such as an outline of a rocket ship) that the user wishes to draw on the display many times, perhaps in different sizes, locations and orientations. Up to 255 different shapes may be created, used, and saved in a "Shape Table", through the use of the High-Resolution subroutines DRAW, DRAW1 and SHLOAD, in conjunction with parameters SHAPE, ROT and SCALE.

In this section, PART D, you will be shown how to create, save and load a Shape Table. The following section, PART E, demonstrates the use of the shape-drawing subroutines with a predefined Shape Table.

HOW TO CREATE A SHAPE TABLE

Before the High-Resolution shape-drawing subroutines can be used, a shape must be defined by a "shape definition." This shape definition consists of a sequence of plotting vectors that are stored in a series of bytes in APPLE's memory. One or more such shape definitions, with their index, make up a "Shape Table" that can be created from the keyboard and saved on disk or cassette tape for future use.

Each byte in a shape definition is divided into three sections, and each section can specify a "plotting vector": whether or not to plot a point, and also a direction to move (up, down, left, or right). The shape-drawing subroutines DRAW and DRAW1 (see PART E) step through each byte in the shape definition section by section, from the definition's first byte through its last byte. When a byte that contains all zeros is reached, the shape definition is complete.

This is how the three sections A, B and C are arranged within one of the bytes that make up a shape definition:

Section:	C		B			A		
Bit Number:	7	6	5	4	3	2	1	Ø
Specifies:	D	D	P	D	D	P	D	D

Each bit pair DD specifies a direction to move, and each bit P specifies whether or not to plot a point before moving, as follows:

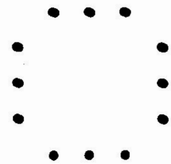
```
If DD = ØØ  move up
      = Ø1  move right          If P = Ø  don't plot
      = 1Ø  move down                = 1  do plot
      = 11  move left
```

Notice that the last section, C (the two most significant bits), does not have a P field (by default, P=Ø), so section C can only specify a move without plotting.

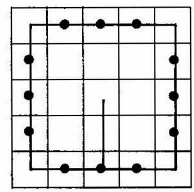
Each byte can represent up to three plotting vectors, one in section A, one in section B, and a third (a move only) in section C.

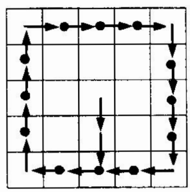
DRAW and DRAW1 process the sections from right to left (least significant bit to most significant bit: section A, then B, then C). At any section in the byte, IF ALL THE REMAINING SECTIONS OF THE BYTE CONTAIN ONLY ZEROS, THEN THOSE SECTIONS ARE IGNORED. Thus, the byte cannot end with a move in section C of ØØ (a move up, without plotting) because that section, containing only zeros, will be ignored. Similarly, if section C is ØØ (ignored), then section B cannot be a move of ØØØ as that will also be ignored. And a move of ØØØ in section A will end your shape definition unless there is a 1-bit somewhere in section B or C.

Suppose you 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 at the center. Next, draw a path through each point in the shape, using only 9Ø degree angles on the turns:





Next, re-draw the shape as a series of plotting vectors, each one moving one place up, down, right, or left, and distinguish the vectors that plot a point before moving (a dot marks vectors that plot points).

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

Next draw a table like the one in Figure 1, below:

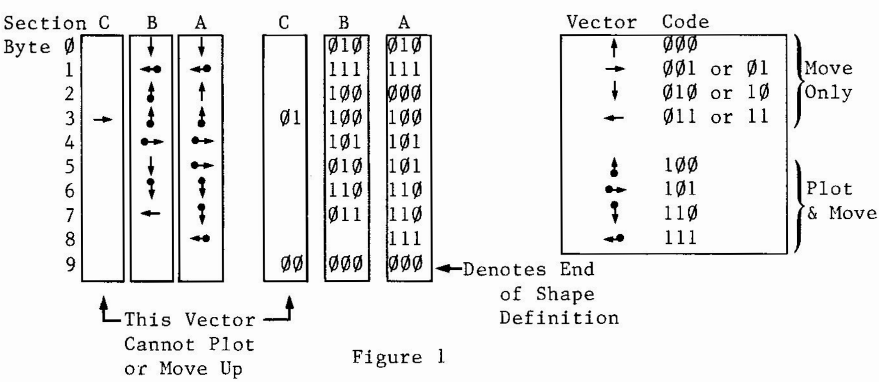


Figure 1

For each vector in the line, determine the bit code and place it in the next available section in the table. If the code will not fit (for example, the vector in section C can't plot a point), or is a ØØ (or ØØØ) at the end of a byte, then skip that section and go on to the next. When you have finished coding all your vectors, check your work to make sure it is accurate.

Now make another table, as shown in Figure 2, below, and re-copy the vector codes from the first table. Recode the vector information into a series of hexadecimal bytes, using the hexadecimal codes from Figure 3.

Section: C (first 2 bits), B (next 3 bits), A (last 3 bits)

Byte	Digit 1	Digit 2		Bytes Recoded in Hex	
Byte Ø	Ø Ø Ø 1	Ø Ø 1 Ø	=	1 2	
1	Ø Ø 1 1	1 1 1 1	=	3 F	
2	Ø Ø 1 Ø	Ø Ø Ø Ø	=	2 Ø	
3	Ø 1 1 Ø	Ø 1 Ø Ø	=	6 4	
4	Ø Ø 1 Ø	1 1 Ø 1	=	2 D	
5	Ø Ø Ø 1	Ø 1 Ø 1	=	1 5	
6	Ø Ø 1 1	Ø 1 1 Ø	=	3 6	
7	Ø Ø Ø 1	1 1 1 Ø	=	1 E	
8	Ø Ø Ø Ø	Ø 1 1 1	=	Ø 7	
9	Ø Ø Ø Ø	Ø Ø Ø Ø	=	Ø Ø	← Denotes End of Shape Definition
Hex:	Digit 1	Digit 2			

Figure 2

Codes

Binary		Hex
ØØØØ	=	Ø
ØØØ1	=	1
ØØ1Ø	=	2
ØØ11	=	3
Ø1ØØ	=	4
Ø1Ø1	=	5
Ø11Ø	=	6
Ø111	=	7
1ØØØ	=	8
1ØØ1	=	9
1Ø1Ø	=	A
1Ø11	=	B
11ØØ	=	C
11Ø1	=	D
111Ø	=	E
1111	=	F

Figure 3

The series of hexadecimal bytes that you arrived at in Figure 2 is the shape definition. There is still a little more information you need to provide before you have a complete Shape Table. The form of the Shape Table, complete with its index, is shown in Figure 4 on the next page.

For this example, your index is easy: there is only one shape definition. The Shape Table's starting location, whose address we have called S, must contain the number of shape definitions (between Ø and 255) in hexadecimal. In this case, that number is just one. We will place our shape definition immediately below the index, for simplicity. That means, in this case, the shape definition will start in byte S+4: the address of shape definition #1, relative to S, is 4 (ØØ Ø4, in hexadecimal). Therefore, index byte S+2 must contain the value Ø4 and index byte S+3 must contain the value ØØ. The completed Shape Table for this example is shown in Figure 5 on the next page.

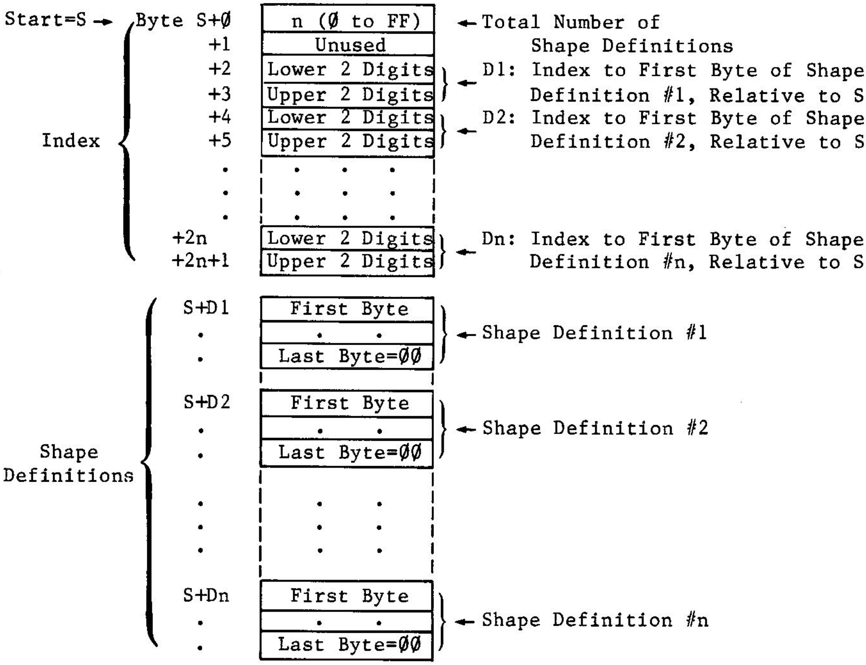


Figure 4

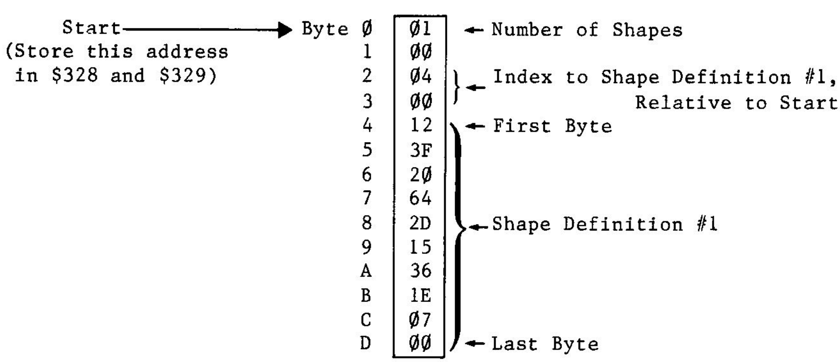


Figure 5

You are now ready to type the Shape Table into APPLE's memory. First, choose a starting address. For this example, we'll use hexadecimal address Ø8ØØ.

Note: this address _must_ be less than the highest memory address available in your system (HIMEM), and not in an area that will be cleared when you use memory Page 1 (hexadecimal locations $2ØØØ to $4ØØØ) or Page 2 (hexadecimal locations $4ØØØ to $6ØØØ) for high-resolution graphics. Furthermore, it must not be in an area of memory used by your BASIC program. Hexadecimal Ø8ØØ (2Ø48, in decimal) is the lowest memory address normally available to a BASIC program. This lowest address is called LOMEM. Later on, we will move the LOMEM pointer higher, to the end of our Shape Table, in order to protect our table from BASIC program variables.

Press the RESET key to enter the Monitor program, and type the Starting address for your Shape Table:

```
 Ø8ØØ
```

If you press the RETURN key now, APPLE will show you the address and the _contents_ of that address. That is how you examine an address to see if you have a put the correct number there. If instead you type a colon (:) followed by a two-digit hexadecimal number, that number will be _stored_ at the specified address when you press the RETURN key. Try this:

```
 Ø8ØØ return
```

(type "return" by pressing the RETURN key). What does APPLE say the contents of location Ø8ØØ are? Now try this:

```
 Ø8ØØ:Ø1 return
 Ø8ØØ return
Ø8ØØ- Ø1
```

The APPLE now says that the value Ø1 (hexadecimal) is stored in the location whose address is Ø8ØØ. To store more two-digit hexadecimal numbers in successive bytes in memory, just open the first address:

```
 Ø8ØØ:
```

and then type the numbers, separated by spaces:

```
 Ø8ØØ:Ø1 ØØ Ø4 ØØ 12 3F 2Ø 64 2D 15 36 1E Ø7 ØØ return
```

You have just typed your first complete Shape Table...not so bad, was it? To check the information in your Shape Table, you can examine each byte separately or simply press the RETURN key repeatedly until all the bytes of interest (and a few extra, probably) have been displayed:

```
 Ø8ØØ return
Ø8ØØ- Ø1
 return
 ØØ Ø4 ØØ 12 3F 2Ø 64
 return
Ø8Ø8- 2D 15 36 1E Ø7 ØØ FF FF
```

If your Shape Table looks correct, all that remains is to store the starting address of the Shape Table where the shape-drawing subroutines can find it (this is done automatically when you use the SHLOAD subroutine to get a table from cassette tape). Your APPLE looks for the four hexadecimal digits of the table's starting address in hexadecimal locations 328 (lower two digits) and 329 (upper two digits). For our table's starting address of Ø8 ØØ, this would do the trick:

```
 328:ØØ Ø8
```

To protect this Shape Table from being erased by the variables in your BASIC program, you must also set LOMEM (the lowest memory address available to your program) to the address that is one byte beyond the Shape Table's last, or largest, address.

It is best to set LOMEM from BASIC, as an immediate-execution command issued before the BASIC program is RUN. LOMEM is automatically set when you invoke BASIC (reset ctrl B return) to decimal 2Ø48 (Ø8ØØ, in hexadecimal). You must then change LOMEM to 2Ø48 plus the number of bytes in your Shape Table plus one. Our Shape Table was decimal 14 bytes long, so our immediate-execution BASIC command would be:

```
LOMEM: 2Ø48 + 15
```

Fortunately, all of this (entering the Shape Table at LOMEM, resetting LOMEM to protect the table, and putting the table's starting address in $328-$329) is taken care of automatically when you use the High-Resolution feature's SHLOAD subroutine to get the table from cassette tape.

SAVING A SHAPE TABLE

Saving on Cassette Tape

To save your Shape Table on tape, you must be in the Monitor and you must know three hexadecimal numbers:

1) Starting Address of the table (Ø8ØØ, in our example)
2) Last Address of the table (Ø8ØD, in our example)
3) Difference between 2) and 1) (ØØØD, in our example)

Item 3, the difference between the last address and the first address of the table, must be stored in hexadecimal locations Ø (lower two digits) and 1 (upper two digits):

```
Ø:ØD ØØ   return
```

Now you can "Write" (store on cassette) first the table length that is stored in locations Ø and 1, and then the Shape Table itself that is stored in locations Starting Address through Last Address:

```
Ø.1W    Ø8ØØ.Ø8ØDW
```

Don't press the RETURN key until you have put a cassette in your tape recorder, rewound it, and started it recording (press PLAY and RECORD simultaneously). Now press the computer's RETURN key.

Saving on Disk

To save your Shape Table on disk, use a command of this form:

```
BSAVE filename, A$ startingaddress, L$ tablelength
```

For our example, you might type

```
BSAVE MYSHAPE1, A$ Ø8ØØ, L$ ØØØD
```

Note: the Disk Operating System (DOS) occupies the top 1Ø.5K of memory (1Ø752 bytes decimal, or $2AØØ hex); make sure your Shape Table is not in that portion of memory when you "boot" the disk system.

LOADING A SHAPE TABLE

Loading from Cassette Tape

To load a Shape Table from cassette tape, rewind the tape, start it playing (press PLAY), and (in BASIC, now) type

```
CALL -11335 return
```

or (if you have previously assigned the value -11335 to the variable SHLOAD)

```
CALL SHLOAD return
```

You should hear one "beep" when the table's length has been read successfully, and another "beep" when the table itself has been read. When loaded this way, your Shape Table will load into memory, beginning at hexadecimal address Ø8ØØ. LOMEM is automatically changed to the address of the location immediately following the last Shape-Table byte. Hexadecimal locations 328 and 329 are automatically set to contain the starting address of the Shape Table.

Loading from Disk

To load a Shape Table from disk, use a command of the form

```
BLOAD filename
```

From our previously-saved example, you would type

```
BLOAD MYSHAPE1
```

This will load your Shape Table into memory, beginning at the address you specified after "A$" when you BSAVEd the Shape Table earlier. In our example, MYSHAPE1 would BLOAD beginning at address Ø8ØØ. You must store the Shape Table's starting address in hexadecimal locations 328 and 329, yourself, from the Monitor:

```
328:ØØ Ø8 return
```

If your Shape Table is in an area of memory that may be used by your BASIC program (as our example is), you must protect the Shape Table from your program. Our example lies at the low end of memory, so we can protect it by raising LOMEM to just above the last byte of the Shape Table. This must be done after invoking BASIC (reset ctrl B return) and before RUNning our BASIC program. We could do this with the immediate-execution BASIC command

```
LOMEM: 2Ø48 + 15
```

FIRST USE OF A SHAPE TABLE

You are now ready to write a BASIC program using Shape-Table subroutines such as DRAW and DRAW1. For a full discussion of these High-Resolution subroutines, see the following section, PART E.

Remember that Page 1 graphics uses memory locations 8192 through 16383 (8K to 16K), and Page 2 graphics uses memory locations 16384 through 24575 (16K to 24K). Integer BASIC puts your program right at the top of available memory; so if your APPLE contains less than 32K of memory, you should protect your program by setting HIMEM to 8192. This must be done after you invoke BASIC (reset ctrl B return) and before RUNning your program, with the immediate-execution command

```
HIMEM: 8192
```

Here's a sample program that assumes our Shape Table has already been loaded from tape, using CALL SHLOAD. This program will print our defined shape, rotate it 5.6 degrees if that rotation is recognized (see ROT discussion, next section) and then repeat, each repetition larger than the one before.

```
10  X0 = Y0 = COLR = SHAPE = ROT = SCALE : REM  SET PARAMETERS
20  INIT = -12288 : DRAW = -11465 : REM  DEFINE SUBROUTINES
30  WHITE = 127 : BLACK = 0 : REM  DEFINE COLORS
40  CALL INIT : REM  INITIALIZE HIGH-RESOLUTION SUBROUTINES
50  SHAPE = 1
60  X0 = 139 : Y0 = 79 : REM  ASSIGN PARAMETER VALUES
70  FOR R = 1 TO 48
80  ROT = R
90  SCALE = R
100  COLR = WHITE
110  CALL DRAW : REM  DRAW SHAPE 1 WITH ABOVE PARAMETERS
120  NEXT R : REM  NEW PARAMETERS
130  END
```

To pause, and then erase each square after it is drawn, add these lines:

```
114  FOR PAUSE = 1 TO 200 : NEXT PAUSE
116  COLR = BLACK : REM  CHANGE COLOR
118  CALL DRAW : REM  RE-DRAW SAME SHAPE, IN NEW COLOR
```

PART E: DRAWING SHAPES FROM A PREPARED SHAPE TABLE

Before either of the two shape-drawing subroutines DRAW or DRAW1 can be used, a "Shape Table" must be defined and stored in memory (see PART E: CREATING A SHAPE TABLE), the Shape Table's starting address must be specified in hexadecimal locations 328 and 329 (8Ø8 and 8Ø9, in decimal), and the High-Resolution subroutines themselves must have been initialized by a CALL INIT.

ASSIGNING PARAMETER VALUES

The DRAW subroutine is used to display any of the shapes defined in the current Shape Table. The origin or 'beginning point' for DRAWing the shape is specified by the values assigned to XØ and YØ, and the rest of the shape continues from that point. The color of the shape to be DRAWn is specified by the value of COLR.

The shape number (the Shape Table's particular shape definition that you wish to have DRAWn) is specified by the value of SHAPE. For example,

```
SHAPE = 3
```

specifies that the next shape-drawing command will use the third shape definition in the Shape Table. SHAPE may be assigned any value (from 1 through 255) that corresponds to one of the shape definitions in the current Shape Table. An attempt to DRAW a shape that does not exist (by executing a shape-drawing command after setting SHAPE = 4, when there are only two shape definitions in your Shape Table, for instance) will result in a *** RANGE ERR message being displayed, and the program will halt.

The relative size of the shape to be DRAWn is specified by the value assigned to SCALE. For example,

```
SCALE = 4
```

specifies that the next shape DRAWn will be four times the size that is described by the appropriate shape definition. That is, each "plotting vector" (either a plot and a move, or just a move) will be repeated four times. SCALE may be assigned any value from Ø through 255, but SCALE = Ø is interpreted as SCALE = 256, the <u>largest</u> size for a given shape definition.

You can also specify the orientation or angle of the shape to be DRAWn, by assigning the proper value to ROT. For example,

```
ROT = Ø
```

will cause the next shape to be DRAWn oriented just as it was defined, while

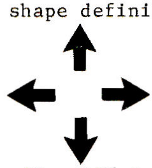
```
ROT = 16
```

will cause the next shape to be DRAWn rotated 9Ø degrees clockwise. The value assigned to ROT must be within the range Ø to 255 (although ROT=64, specifying a rotation of 36Ø degrees clockwise, is the equivalent of ROT=Ø). For SCALE=1, only four of the 63 different rotations are recognized (Ø,16,32,48); for SCALE=2, eight different rotations are recognized; etc. ROT values specifying unrecognized rotations will usually cause the shape to be DRAWn with the next smaller recognized rotation.

ORIENTATIONS OF SHAPE DEFINITION

ROT = Ø (no rotation from shape definition)

ROT = 48 (27Ø degrees clockwise rotation)



ROT = 16 (9Ø degrees clockwise rotation)

ROT = 32 (18Ø degrees clockwise rotation)

DRAWING SHAPES

The following example program DRAWs shape definition number three, in white, at a 135 degree clockwise rotation. Its starting point, or origin, is at (14Ø,8Ø).

```
Ø  XØ = YØ = COLR = SHAPE = ROT = SCALE : REM  SET PARAMETERS
5  INIT = -12288 : DRAW = -11465 : REM  DEFINE SUBROUTINES
1Ø  WHITE = 127 : REM  DEFINE COLOR
2Ø  CALL INIT : REM  INITIALIZE HIGH-RESOLUTION SUBROUTINES
3Ø  XØ = 14Ø : YØ = 8Ø : COLR = WHITE : REM  ASSIGN PARAMETER VALUES
4Ø  SHAPE = 3 : ROT = 24 : SCALE = 2
5Ø  CALL DRAW : REM  DRAW SHAPE 3, DOUBLE SIZE, TURNED 135 DEGREES
6Ø  END
```

LINKING SHAPES

DRAW1 is identical to DRAW, except that the last point previously DRAWn, PLOTted or POSNed determines the color and the starting point for the new shape. XØ, YØ, and COLR, need not be specified, as they will have no effect on DRAW1. However, some point must have been plotted before CALLing DRAW1, or this CALL will have no effect.

The following example program draws "squiggles" by DRAWing a small shape whose orientation is given by game control #Ø, then linking a new shape to the old one, each time the game control gives a new orientation. To clear the screen of "squiggles," press the game-control button.

```
1Ø  XØ = YØ = COLR = SHAPE = ROT = SCALE : REM  SET PARAMETERS
2Ø  INIT = -12288 : DRAW = -11465 : DRAW1 = -11462
22  CLEAR = -12274 : WHITE = 127 : REM  NAME SUBROUTINES AND COLOR
3Ø  FULLSCREEN = -163Ø2 : BUTN = -16287 : REM  NAME LOCATIONS
4Ø  CALL INIT : REM  INITIALIZE HIGH-RESOLUTION SUBROUTINES
5Ø  POKE FULLSCREEN, Ø : REM  SET FULL-SCREEN GRAPHICS
6Ø  COLR = WHITE : SHAPE = 1 : SCALE = 5
7Ø  XØ = 14Ø : YØ = 8Ø : REM  ASSIGN PARAMETER VALUES
8Ø  CALL CLEAR : ROT = PDL(Ø) : CALL DRAW : REM  DRAW FIRST SHAPE
9Ø  IF PEEK(BUTN) > 127 THEN GOTO 8Ø : REM  PRESS BUTTON TO CLEAR SCREEN
1ØØ  R = PDL(Ø) : IF (R < ROT+2) AND (R > ROT-2) THEN GOTO 9Ø :
     REM  WAIT FOR CHANGE IN GAME CONTROL
11Ø  ROT = R : CALL DRAW1 : REM  ADD TO "SQUIGGLE"
12Ø  GOTO 9Ø : REM  LOOK FOR ANOTHER CHANGE
```

After DRAWing a shape, you may wish to draw a LINE from the last plotted point of the shape to another fixed point on the screen. To do this, once the shape is DRAWn, you must first use

```
CALL FIND
```

prior to CALLing LINE. The FIND subroutine determines the X and Y coordinates of the final point in the shape that was DRAWn, and uses it as the beginning point for the subsequent CALL LINE.

The following example DRAWs a white shape, and then draws a violet LINE from the final plot position of the shape to the point (1Ø, 25).

```
Ø  XØ = YØ = COLR = SHAPE = ROT = SCALE : REM  SET PARAMETERS
5  INIT = -12288 : LINE = -115ØØ : DRAW = -114Ø2 : FIND = -1178Ø
1Ø  VIOLET = 85 : WHITE = 127 : REM  DEFINE SUBROUTINES AND COLORS
2Ø  XØ = 14Ø : YØ = 8Ø : COLR = WHITE : REM  ASSIGN PARAMETER VALUES
3Ø  SHAPE = 3 : ROT = Ø : SCALE = 2
4Ø  CALL DRAW : REM  DRAW SHAPE WITH ABOVE PARAMETERS
5Ø  CALL FIND : REM  FIND COORDINATES OF LAST SHAPE POINT
6Ø  XØ = 1Ø : YØ = 25 : COLR = VIOLET : REM  NEW PARAMETER VALUES, FOR LINE
7Ø  CALL LINE : REM  DRAW LINE WITH ABOVE PARAMETERS
8Ø  END
```

COLLISIONS

Any time two or more shapes intersect or overlap, the new shape has points in common with the previous shapes. These common points are called points of "collision."

The DRAW and DRAW1 subroutines return a "collision count" in the hexadecimal memory location $32A (81Ø, in decimal). The collision count will be constant for a fixed shape, rotation, scale, and background, provided that no collisions with other shapes are detected. The difference between the "standard" collision value and the value encountered while DRAWing a shape is a true collision counter. For example, the collision counter is useful for determining whether or not two constantly moving shapes ever touch each other.

```
11Ø  CALL DRAW : REM DRAW THE SHAPE
12Ø  COUNT = PEEK(81Ø) : REM FIND THE COLLISION COUNT
```

PART F: TECHNICAL INFORMATION

LOCATIONS OF THE HIGH-RESOLUTION PARAMETERS

When the high-resolution parameters are entered (line Ø, say), they are stored -- with space for their values -- in the BASIC variable table, just above LOMEM (the LOwest MEMory location used for BASIC variable storage). These parameters appear in the variable table in the exact order of their first mention in the BASIC program. That order must be as shown below, because the High-Resolution subroutines look for the parameter values by location only. Each parameter value is two bytes in length. The low-order byte is stored in the lesser of the two locations assigned.

VARIABLE-TABLE PARAMETER LOCATIONS

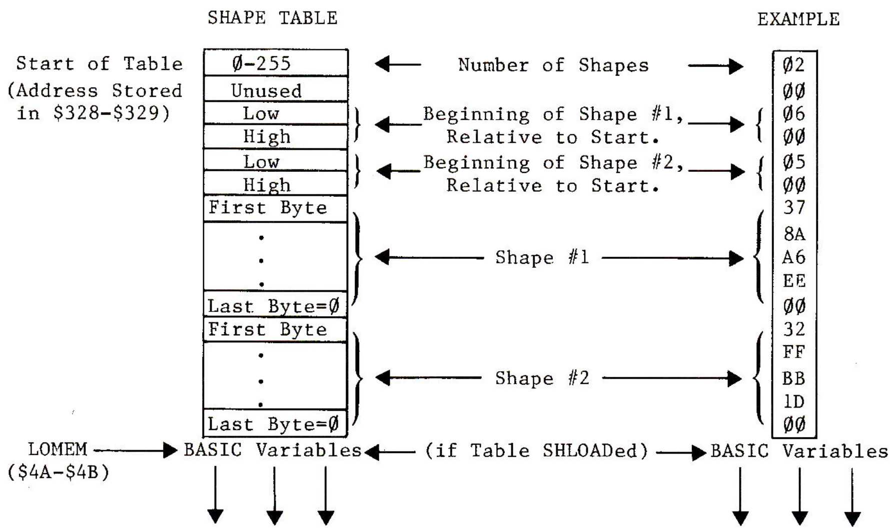
Parameter	Locations beyond LOMEM
XØ	$Ø5, $Ø6
YØ	$ØC, $ØD
COLR	$15, $16
SHAPE	$1F, $2Ø
ROT	$27, $28
SCALE	$31, $32

VARIABLES USED WITHIN THE HIGH-RESOLUTION SUBROUTINES

Variable Name	Hexadecimal Location	Description
SHAPEL, SHAPEH	1A, 1B	On-the-fly shape pointer.
HCOLOR1	1C	On-the-fly color byte.
COUNTH	1D	High-order byte of step count for LINE.
HBASL, HBASH	26, 27	On-the-fly BASE ADDRESS
HMASK	3Ø	On-the-fly BIT MASK
QDRNT	53	2 LSB's are rotation quadrant for DRAW.
XOL, XOH	32Ø, 321	Most recent X-coordinate. Used for initial endpoint of LINE. Updated by PLOT, POSN, LINE and FIND, not DRAW.
YO	322	Most recent Y-coordinate (see XOL, XOH).
BXSAV	323	Saves 65Ø2 X-register during high-resolution CALLs from BASIC.
HCOLOR	324	Color specification for PLOT, POSN.
HNDX	325	On-the-fly byte index from BASE ADDRESS.
HPAG	326	Memory page for plotting graphics. Normally $2Ø for plotting in Page 1 of high-resolution display memory ($2ØØØ-$3FFF).
SCALE	327	On-the-fly scale factor for DRAW.
SHAPXL, SHAPXH	328, 329	Start of Shape Table pointer.
COLLSN	32A	Collision count from DRAW, DRAW1.

SHAPE TABLE INFORMATION

Shape Tape	Description
Record #1	A two-byte-long record that contains the length of record #2, Low-order first.
Record Gap	Minumum of .7 seconds in length.
Record #2	The Shape Table (see below).

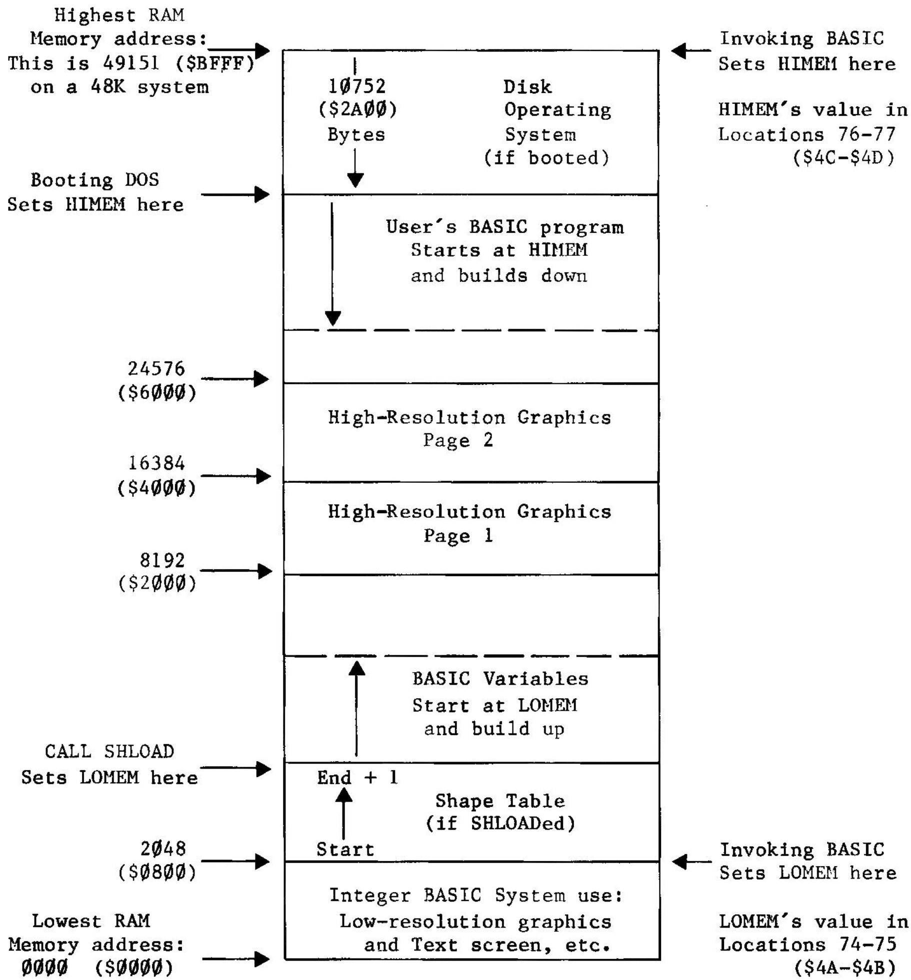


The address of the Shape Table's Start should be stored in locations $328 and $329. If the SHLOAD subroutine is used to load the table, Start will be set to LOMEM (normally this is at $Ø8ØØ) and then LOMEM will be moved to one byte after the end of the Shape Table, automatically.

If you wish to load a Shape Table named MYSHAPES2 from disk, beginning at decimal location 2Ø48 (Ø8ØØ hex), and ending at decimal location 2Ø48 plus decimal 15 bytes (as in the example above), you may wish to begin your BASIC program as follows:

```
Ø  D$ = "" : REM  QUOTES CONTAIN CTRL D (D$ WILL BE ERASED BY SHAPE TABLE)
1  PRINT D$; "BLOAD MYSHAPES2 , A 2Ø48" : REM  LOADS SHAPE TABLE
2  POKE 8Ø8, 2Ø48 MOD 256 : POKE 8Ø9, 2Ø48 / 256 : REM  SETS TABLE START
3  POKE 74, (2Ø48 + 15 + 1) MOD 256 : POKE 75, (2Ø48 + 15 + 1) / 256
4  POKE 2Ø4, PEEK(74) : POKE 2Ø5, PEEK(75) : REM  SETS LOMEM TO TABLE END+1
5  XØ = YØ = COLR = SHAPE = ROT = SCALE : REM  SETS PARAMETERS
```

APPLE II MEMORY MAP FOR USING HIGH-RESOLUTION GRAPHICS WITH INTEGER BASIC



Unfortunately, there is no convention for mapping memory. This map shows the highest (largest) address at the top, lowest (smallest) address at the bottom. The maps of Shape Tables that appear on other pages show the Starting address (lowest and smallest) at the top, the Ending address (highest and largest) at the bottom.

PART G: COMMENTS

1. Using memory Page 1 for high-resolution graphics erases everything in memory from location 8192 ($2000 hex) to location 16383 ($3FFF). If the top of your system's memory is in this range (as it will be, if you have a 16K system), Integer BASIC will normally put your BASIC program exactly where it will be erased by INIT. You must protect your program by setting HIMEM below memory Page 1, after invoking BASIC (reset ctrl B return) and before RUNning your program: use this immediate-execution command:

HIMEM: 8192 return

2. Using memory Page 2 for high-resolution graphics erases memory from location 16384 ($4000) to location 24575 ($5FFF). If yours is a 24K system, this will erase your BASIC program unless you do one of the following:

a) never use Page 2 for graphics; or
b) change HIMEM to 8192, as described above.

3. The picture is further confused if you are also using an APPLE disk with your system. The Disk Operating System (DOS), when booted, occupies the highest 10.5K ($2A00) bytes of memory. HIMEM is moved to just below the DOS. Therefore, if your system contains less than 32K of memory, the DOS will occupy memory Page 1 and Page 2. In that case, you cannot use the High-Resolution graphics with the DOS intact. An attempt to do so will erase all or part of the DOS. A 32K system can use only Page 1 for graphics without destroying the DOS, but HIMEM must be moved to location 8192 as described above. 48K systems can usually use the DOS and both high-resolution memory pages without problems.

4. If you loaded your Shape Table starting at LOMEM in location 2048 ($0800), from disk or from tape without using SHLOAD, Integer BASIC will erase the Shape Table when it stores the program variables. To protect your Shape Table, you must move LOMEM to one byte beyond the last byte of the Shape Table, after invoking BASIC and before using <u>any</u> variables. SHLOAD does this automatically, but you can use this immediate-execution command:

LOMEM: 2048 + tablelength + 1

where tablelength must be a number, <u>not</u> a variable name. Some programmers load their Shape Tables beginning in location 3048 ($0BE8). That leaves a safe margin of 1000 bytes for variables below the Shape Table, and at least 5000 bytes (if HIMEM:8192) above the table for their BASIC program.

5. CALLing an undefined or accidentally misspelled variable name is usually a CALL to location zero (the default value of any undefined variable). This CALL may cause unpredictable and unwelcome results, depending on the contents of location zero. However, after you execute this BASIC command:

POKE 0, 96

an accidental CALL to location zero will cause a simple jump back to your BASIC program, with no damage.

APPENDIX I
SOURCE ASSEMBLY LISTINGS

```
          ***************************
          *                         *
          * APPLE-II HI-RESOLUTION  *
          *  GRAPHICS SUBROUTINES   *
          *                         *
          *    BY WOZ   9/13/77     *
          *                         *
          *  ALL RIGHTS RESERVED    *
          *                         *
          ***************************

          *  HI-RES EQUATES
          SHAPEL  EQU   $1A POINTER TO
          SHAPEH  EQU   $1B SHAPE LIST
          HCOLOR1 EQU   $1C RUNNING COLOR MASK
          COUNTH  EQU   $1D
          HBASL   EQU   $26 BASE ADR FOR CURRENT
          HBASH   EQU   $27  HI-RES PLOT LINE. A
          HMASK   EQU   $30
          A1L     EQU   $3C MONITOR A1.
          A1H     EQU   $3D
          A2L     EQU   $3E MONITOR A2.
          A2H     EQU   $3F
          LOMEML  EQU   $4A BASIC 'START OF VARS'.
          LOMEMH  EQU   $4B
          DXL     EQU   $50 DELTA-X FOR HLIN, SHAPE.
          DXH     EQU   $51
          SHAPEX  EQU   $51 SHAPE TEMP.
          DY      EQU   $52 DELTA-Y FOR HLIN, SHAPE.
          QDRNT   EQU   $53 ROT QUADRANT (SHAPE).
          EL      EQU   $54 ERROR FOR HLIN.
          EH      EQU   $55
          PPL     EQU   $CA BASIC START OF PROG PTR.
          PPH     EQU   $CB
          PVL     EQU   $CC BASIC END OF VARS PTR.
          PVH     EQU   $CD
          ACL     EQU   $CE BASIC ACC.
          ACH     EQU   $CF
          XOL     EQU   $320 PRIOR X-COORD SAVE
          XOH     EQU   $321 AFTER HLIN OR HPLOT.
          YO      EQU   $322 HLIN, HPLOT Y-COORD SAVE.
          BXSAV   EQU   $323 X-REG SAVE FOR BASIC.
          HCOLOR  EQU   $324 COLOR FOR HPLOT, HPOSN
          HNDX    EQU   $325 HORIZ OFFSET SAVE.
          HPAG    EQU   $326 HI-RES PAGE ($20 NORMAL)
          SCALE   EQU   $327 SCALE FOR SHAPE, MOVE.
          SHAPXL  EQU   $328 START OF
          SHAPXH  EQU   $329   SHAPE TABLE.
          COLLSN  EQU   $32A COLLISION COUNT.
          HIRES   EQU   $C057 SWITCH TO HI-RES VIDEO
          MIXSET  EQU   $C053 SELECT TEXT/GRAPHICS MIX
          TXTCLR  EQU   $C050 SELECT GRAPHICS MODE.
          MEMFUL  EQU   $E36B BASIC MEM FULL ERROR.
          RNGERR  EQU   $EE68 BASIC RANGE ERROR.
          ACADR   EQU   $F11E 2-BYTE TAPE READ SETUP.
          RD2BIT  EQU   $FCFA TWO-EDGE TAPE SENSE.
          READ    EQU   $FEFD TAPE READ (A1.A2).
          READX1  EQU   $FF02 READ WITHOUT HEADER.

          *  HIGH RESOLUTION GRAPHICS INITS
          *
          * ROM VERSION $D000 TO $D3FF
          *
                  ORG   $D000
                  OBJ   $A000
D000 A9 20        SETHRL LDA   #$20 INIT FOR $2000-3FFF
D002 8D 26 03             STA   HPAG HI-RES SCREEN MEMORY.
```

```
D005 AD 57 C0            LDA   HIRES SET HIRES DISPLAY MODE
D008 AD 53 C0            LDA   MIXSET WITH TEXT AT BOTTOM.
D00B AD 50 C0            LDA   TXTCLR SET GRAPHICS DISPLAY MODE
D00E A9 00      HCLR     LDA   #$0
D010 85 1C      BKGND0   STA   HCOLOR1 SET FOR BLACK BKGND.
D012 AD 26 03   BKGND    LDA   HPAG
D015 85 1B               STA   SHAPEH INIT HI-RES SCREEN MEM
D017 A0 00               LDY   #$0 FOR CURRENT PAGE, NORMALLY
D019 84 1A               STY   SHAPEL $2000-3FFF OR $4000-5FFF
D01B A5 1C      BKGND1   LDA   HCOLOR1
D01D 91 1A               STA   (SHAPEL),Y
D01F 20 A2 D0            JSR   CSHFT2 (SHAPEL,H) WILL SPECIFY
D022 C8                  INY   32 SEPARATE PAGES.
D023 D0 F6               BNE   BKGND1 THROUGHOUT THE INIT.
D025 E6 1B               INC   SHAPEH
D027 A5 1B               LDA   SHAPEH
D029 29 1F               AND   #$1F TEST FOR DONE.
D02B D0 EE               BNE   BKGND1
D02D 60                  RTS

                *  HI-RES GRAPHICS POSITION AND PLOT SUBRS
D02E 8D 22 03   HPOSN    STA   Y0 ENTER WITH Y IN A-REG,
D031 8E 20 03            STX   X0L XL IN X-REG,
D034 8C 21 03            STY   X0H AND XH IN Y-REG.
D037 48                  PHA
D038 29 C0               AND   #$C0
D03A 85 26               STA   HBASL FOR Y-COORD = 00ABCDEF.
D03C 4A                  LSR   ;CALCULATES BASE ADDRESS
D03D 4A                  LSR   ;IN HBASL, HBASH FOR
D03E 05 26               ORA   HBASL    ACCESSING SCREEN MEM
D040 85 26               STA   HBASL    VIA (HBASL),Y ADDRESSING MODE
D042 68                  PLA
D043 85 27               STA   HBASH
D045 0A                  ASL   ;CALCULATES
D046 0A                  ASL   ;HBASH = PPPFGHCD,
D047 0A                  ASL   ;HBASL = EABAB000
D048 26 27               ROL   HBASH
D04A 0A                  ASL   ;WHERE PPP=001 FOR $2000-3FFF
D04B 26 27               ROL   HBASH SCREEN MEM RANGE AND
D04D 0A                  ASL   ; PPP=010 FOR $4000-7FFF
D04E 66 26               ROR   HBASL (GIVEN Y-COORD=ABCDEFGH)
D050 A5 27               LDA   HBASH
D052 29 1F               AND   #$1F
D054 0D 26 03            ORA   HPAG
D057 85 27               STA   HBASH
D059 8A                  TXA   DIVIDE X0 BY 7 FOR
D05A C0 00               CPY   #$0 INDEX FROM BASE ADR
D05C F0 05               BEQ   HPOSN2 (QUOTIENT) AND BIT
D05E A0 23               LDY   #$23 WITHIN SCREEN MEM BYTE
D060 69 04               ADC   #$4 (MASK SPEC'D BY REMAINDER)
D062 C8         HPOSN1   INY
D063 E9 07      HPOSN2   SBC   #$7 SUBTRACT OUT SEVENS.
D065 B0 FB               BCS   HPOSN1
D067 8C 25 03            STY   HNDX WORKS FOR X0 FROM
D06A AA                  TAX   0 TO 279, LOW-ORDER
D06B BD EA D0            LDA   MSKTBL-249,X BYTE IN X-REG,
D06E 85 30               STA   HMASK HIGH IN Y-REG ON ENTRY
D070 98                  TYA
D071 4A                  LSR   ; IF ON ODD BYTE (CARRY SET)
D072 AD 24 03            LDA   HCOLOR THEN ROTATE HCOLOR ONE
D075 85 1C      HPOSN3   STA   HCOLOR1 BIT FOR 180 DEGREE SHIFT
D077 B0 29               BCS   CSHFT2 PRIOR TO COPYING TO HCOLOR1.
D079 60                  RTS
D07A 20 2E D0   HPLOT    JSR   HPOSN
D07D A5 1C      HPLOT1   LDA   HCOLOR1 CALC BIT POSN IN HBASL,H
D07F 51 26               EOR   (HBASL),Y HNDX, AND HMASK FROM
D081 25 30               AND   HMASK Y-COORD IN A-REG,
D083 51 26               EOR   (HBASL),Y X-COORD IN X,Y-REGS.
D085 91 26               STA   (HBASL),Y FOR ANY 'L' BITS OF HMASK
D087 60                  RTS   SUBSTITUTE CORRESPONDING
                *                    BIT OF HCOLOR1.
```

```
                 *  HI-RES GRAPHICS L,R,U,D SUBRS
D038 10 24       LFTRT  BPL   RIGHT USE SIGN FOR LFT/RT SELECT
D03A A5 30       LEFT   LDA   HMASK
D03C 4A                 LSR   ; SHIFT LOW-ORDER
D03D B0 05              BCS   LEFT1 7 BITS OF HMASK
D03F 49 C0              EOR   #$C0 ONE BIT TO LSB.
D091 85 30       LR1    STA   HMASK
D093 60                 RTS
D094 88          LEFT1  DEY   DECR HORIZ INDEX.
D095 10 02              BPL   LEFT2
D097 A0 27              LDY   #$27 WRAP AROUND SCREEN.
D099 A9 C0       LEFT2  LDA   #$C0 NEW HMASK, RIGHTMOST
D09B 85 30       NEWNDX STA   HMASK DOT OF BYTE.
D09D 8C 25 03           STY   HNDX UPDATE HORIZ INDEX.
D0A0 A5 1C       CSHIFT LDA   HCOLOR1
D0A2 0A          CSHFT2 ASL   ; ROTATE LOW-ORDER
D0A3 C9 C0              CMP   #$C0 7 BITS OF HCOLOR1
D0A5 10 06              BPL   RTS1 ONE BIT POSN.
D0A7 A5 1C              LDA   HCOLOR1
D0A9 49 7F              EOR   #$7F ZXYXYXYX -> ZYXYXYXY
D0AB 85 1C              STA   HCOLOR1
D0AD 60          RTS1   RTS
D0AE A5 30       RIGHT  LDA   HMASK
D0B0 0A                 ASL   ; SHIFT LOW-ORDER
D0B1 49 80              EOR   #$80 7 BITS OF HMASK
D0B3 30 DC              BMI   LR1 ONE BIT TO MSB.
D0B5 A9 81              LDA   #$81
D0B7 C8                 INY   NEXT BYTE.
D0B8 C0 28              CPY   #$28
D0BA 90 DF              BCC   NEWNDX
D0BC A0 00              LDY   #$0 WRAP AROUND SCREEN IF >279
D0BE B0 DB              BCS   NEWNDX ALWAYS TAKEN.

                 *  L,R,U,D, SUBROUTINES.
D0C0 18          LRUDX1 CLC   NO 90 DEG ROT (X-OR).
D0C1 A5 51       LRUDX2 LDA   SHAPEX
D0C3 29 04              AND   #$4 IF B2=0 THEN NO PLOT.
D0C5 F0 27              BEQ   LRUD4
D0C7 A9 7F              LDA   #$7F FOR EX-OR INTO SCREEN MEM
D0C9 25 30              AND   HMASK
D0CB 31 26              AND   (HBASL),Y SCREEN BIT SET?
D0CD D0 1B              BNE   LRUD3
D0CF EE 2A 03           INC   COLLSN
D0D2 A9 7F              LDA   #$7F
D0D4 25 30              AND   HMASK
D0D6 10 12              BPL   LRUD3 ALWAYS TAKEN.
D0D8 18          LRUD1  CLC   NO 90 DEG ROT.
D0D9 A5 51       LRUD2  LDA   SHAPEX
D0DB 29 04              AND   #$4 IF B2=0 THEN NO PLOT.
D0DD F0 0F              BEQ   LRUD4
D0DF B1 26              LDA   (HBASL),Y
D0E1 45 1C              EOR   HCOLOR1 SET HI-RES SCREEN BIT
D0E3 25 30              AND   HMASK TO CORRESPONDING HCOLOR1
D0E5 D0 03              BNE   LRUD3 IF BIT OF SCREEN CHANGES
D0E7 EE 2A 03           INC   COLLSN THEN INCR COLLSN DETECT
D0EA 51 26       LRUD3  EOR   (HBASL),Y
D0EC 91 26              STA   (HBASL),Y
D0EE A5 51       LRUD4  LDA   SHAPEX ADD QDRNT TO
D0F0 65 53              ADC   QDRNT SPECIFIED VECTOR
D0F2 29 03              AND   #$3 AND MOVE LFT, RT,
                 EQ3    EQU   *-1 UP, OR DWN BASED
D0F4 C9 02              CMP   #$2 ON SIGN AND CARRY.
D0F6 6A                 ROR
D0F7 B0 8F       LRUD   BCS   LFTRT
D0F9 30 30       UPDWN  BMI   DOWN4 SIGN FOR UP/DWN SELECT
D0FB 18          UP     CLC
D0FC A5 27              LDA   HBASH CALC BASE ADDRESS
D0FE 2C EA D1           BIT   EQ1C (ADR OF LEFTMOST BYTE)
D101 D0 22              BNE   UP4 FOR NEXT LINE UP
D103 06 26              ASL   HBASL IN (HBASL, HBASH)
```

```
D105 B0 1A               BCS   UP2 WITH 192-LINE WRAPAROUND
D107 2C F3 D0            BIT   EQ3
D10A F0 05               BEQ   UP1
D10C 69 1F               ADC   #$1F **** BIT MAP ****
D10E 38                  SEC
D10F B0 12               BCS   UP3 FOR ROW = ABCDEFGH,
D111 69 23      UP1      ADC   #$23
D113 48                  PHA
D114 A5 26               LDA   HBASL HBASL = EABAB000
D116 69 B0               ADC   #$B0 HBASH = PPPFGHCD
D118 B0 02               BCS   UP5
D11A 69 F0               ADC   #$F0 WHERE PPP=001 FOR PRIMARY
D11C 85 26      UP5      STA   HBASL HI-RES PAGE ($2000-$3FFF)
D11E 68                  PLA
D11F B0 02               BCS   UP3
D121 69 1F      UP2      ADC   #$1F
D123 66 26      UP3      ROR   HBASL
D125 69 FC      UP4      ADC   #$FC
D127 85 27      UPDWN1   STA   HBASH
D129 60                  RTS
D12A 18         DOWN     CLC
D12B A5 27      DOWN4    LDA   HBASH
D12D 69 04               ADC   #$4 CALC BASE ADR FOR NEXT LINE
                EQ4      EQU   *-1 DOWN TO (HBASL,HBASH)
D12F 2C EA D1            BIT   EQ1C
D132 D0 F3               BNE   UPDWN1

D134 06 26               ASL   HBASL WITH 192-LINE WRAPAROUND
D136 90 19               BCC   DOWN1
D138 69 E0               ADC   #$E0
D13A 18                  CLC
D13B 2C 2E D1            BIT   EQ4
D13E F0 13               BEQ   DOWN2
D140 A5 26               LDA   HBASL
D142 69 50               ADC   #$50
D144 49 F0               EOR   #$F0
D146 F0 02               BEQ   DOWN3
D148 49 F0               EOR   #$F0
D14A 85 26      DOWN3    STA   HBASL
D14C AD 26 03            LDA   HPAG
D14F 90 02               BCC   DOWN2
D151 69 E0      DOWN1    ADC   #$E0
D153 66 26      DOWN2    ROR   HBASL
D155 90 D0               BCC   UPDWN1

                *  HI-RES GRAPHICS LINE DRAW SUBRS
D157 48         HLINRL   PHA
D158 A9 00               LDA   #$0 SET (XOL,XOH) AND
D15A 8D 20 03            STA   XOL YO TO ZERO FOR
D15D 8D 21 03            STA   XOH REL LINE DRAW
D160 8D 22 03            STA   YO (DX, DY).
D163 68                  PLA
D164 48         HLIN     PHA   ON ENTRY
D165 38                  SEC   XL: A-REG
D166 ED 20 03            SBC   XOL XH; X-REG
D169 48                  PHA   Y: Y-REG
D16A 8A                  TXA
D16B ED 21 03            SBC   XOH
D16E 85 53               STA   QDRNT CALC ABS(X-XO)
D170 B0 0A               BCS   HLIN2 IN (DXL,DXH)
```

```
D172 68                  PLA
D173 49 FF               EOR    #$FF X DIR TO SIGN BIT
D175 69 01               ADC    #$1 OF QDRNT.
D177 48                  PHA    0=RIGHT (DX POS)
D178 A9 00               LDA    #$0 1=LEFT (DX NEG)
D17A E5 53               SBC    QDRNT
D17C 85 51       HLIN2   STA    DXH
D17E 85 55               STA    EH INIT (EL,EH) TO
D180 68                  PLA    ABS(X-XO)
D181 85 50               STA    DXL
D183 85 54               STA    EL
D185 68                  PLA
D186 8D 20 03            STA    XOL
D189 8E 21 03            STX    XOH
D18C 98                  TYA
D18D 18                  CLC
D18E ED 22 03            SBC    YO CALC -ABS(Y-O)-1
D191 90 04               BCC    HLIN3 IN DY.
D193 49 FF               EOR    #$FF
D195 69 FE               ADC    #$FE
D197 85 52       HLIN3   STA    DY ROTATE Y DIR INTO
D199 8C 22 03            STY    YO QDRNT SIGN BIT
D19C 66 53               ROR    QDRNT (0=UP, 1=DOWN)
D19E 38                  SEC
D19F E5 50               SBC    DXL INIT (COUNTL, COUNTH).
D1A1 AA                  TAX    TO -(DELTX+DELTY+1)
D1A2 A9 FF               LDA    #$FF
D1A4 E5 51               SBC    DXH
D1A6 85 1D               STA    COUNTH
D1A8 AC 25 03            LDY    HNDX HORIZ INDEX
D1AB B0 05               BCS    MOVEX2 ALWAYS TAKEN.
D1AD 0A          MOVEX   ASL    ; MOVE IN X-DIR. USE
D1AE 20 88 D0            JSR    LFTRT QDRNT B6 FOR LFT/RT SELECT
D1B1 38                  SEC
D1B2 A5 54       MOVEX2  LDA    EL ASSUME CARRY SET.
D1B4 65 52               ADC    DY (EL,EH)-DELTY TO (EL,EH)
D1B6 85 54               STA    EL NOTE: DY IS (-DELTY)-1
D1B8 A5 55               LDA    EH CARRY CLR IF (EL,EH)
D1BA E9 00               SBC    #$0 GOES NEG.
D1BC 85 55       HCOUNT  STA    EH
D1BE B1 26               LDA    (HBASL),Y SCREEN BYTE.
D1C0 45 1C               EOR    HCOLOR1 PLOT DOT OF HCOLOR1.
D1C2 25 30               AND    HMASK CURRENT BIT MASK.
D1C4 51 26               EOR    (HBASL),Y
D1C6 91 26               STA    (HBASL),Y
D1C8 E8                  INX    DONE (DELTX+DELTY)
D1C9 D0 04               BNE    HLIN4 DOTS?
D1CB E6 1D               INC    COUNTH
D1CD F0 6B               BEQ    RTS2 YES, RETURN.
D1CF A5 53       HLIN4   LDA    QDRNT FOR DIRECTION TEST
D1D1 B0 DA               BCS    MOVEX IF CAR SET, (EL, EH) POS
D1D3 20 F9 D0            JSR    UPDWN IF CLR, NEG, MOVE YDIR
D1D6 18                  CLC
D1D7 A5 54               LDA    EL (EL,EH)+DELTX
D1D9 65 50               ADC    DXL TO (EL,EH).
D1DB 85 54               STA    EL
D1DD A5 55               LDA    EH CAR SET IF (EL,EH) GOES POS
D1DF 65 51               ADC    DXH
D1E1 50 D9               BVC    HCOUNT ALWAYS TAKEN.
D1E3 81          MSKTBL  HEX    81 LEFTMOST BIT OF BYTE.
D1E4 82 84 88            HEX    82,84,88
D1E7 90 A0               HEX    90,A0
D1E9 C0                  HEX    C0 RIGHTMOST BIT OF BYTE.
D1EA 1C          EQ1C    HEX    1C
D1EB FF FE FA    COS     HEX    FF,FE,FA,F4,EC,E1,D4,C5,B4
D1F4 A1 8D 78            HEX    A1,8D,78,61,49,31,18,FF
```

```
                   *  HI-RES GRAPHICS COORDINATE RESTORE SUBR
D1FC A5 26          HFIND  LDA   HBASL
D1FE 0A                    ASL   ; CONVERTS BASE ADR
D1FF A5 27                 LDA   HBASH TO Y-COORD.
D201 29 03                 AND   #$3
D203 2A                    ROL   ; FOR HBASL = EABAB000
D204 05 26                 ORA   HBASL HBASH = PPPFGHCD
D206 0A                    ASL
D207 0A                    ASL   ; GENERATE
D208 0A                    ASL   ; Y-COORD = ABCDEFGH
D209 8D 22 03              STA   Y0
D20C A5 27                 LDA   HBASH (PPP=SCREEN PAGE,
D20E 4A                    LSR   ; NORMALLY 001 FOR
D20F 4A                    LSR   ; $2000-$3FFF
D210 29 07                 AND   #$7 HI-RES SCREEN)
D212 0D 22 03              ORA   Y0
D215 8D 22 03              STA   Y0 CONVERTS HNDX (INDEX
D218 AD 25 03              LDA   HNDX FROM BASE ADR)
D21B 0A                    ASL   ; AND HMASK (BIT
D21C 6D 25 03              ADC   HNDX MASK) TO X-COORD
D21F 0A                    ASL   ; IN (X0L,X0H)
D220 AA                    TAX   (RANGE $0-$133)
D221 CA                    DEX
D222 A5 30                 LDA   HMASK
D224 29 7F                 AND   #$7F
D226 E8             HFIND1 INX
D227 4A                    LSR
D228 D0 FC                 BNE   HFIND1
D22A 8D 21 03              STA   X0H
D22D 8A                    TXA
D22E 18                    CLC   CALC HNDX*7 +
D22F 6D 25 03              ADC   HNDX LOG (BASE 2) HMASK.
D232 90 03                 BCC   HFIND2
D234 EE 21 03              INC   X0H
D237 8D 20 03       HFIND2 STA   X0L
D23A 60             RTS2   RTS

                   *  HI-RES GRAPHICS SHAPE DRAW SUBR
                   *
                   * SHAPE DRAW
                   * R = 0 TO 63
                   * SCALE FACTOR USED (1=NORMAL)
                   *
D23B 86 1A          DRAW   STX   SHAPEL DRAW DEFINITION
D23D 84 1B                 STY   SHAPEH POINTER.
D23F AA             DRAW1  TAX
D240 4A                    LSR   ; ROT ($0-$3F)
D241 4A                    LSR
D242 4A                    LSR   ; QDRNT 0=UP, 1=RT,
D243 4A                    LSR   ; 2=DWN, 3=LFT.
D244 85 53                 STA   QDRNT
D246 8A                    TXA
D247 29 0F                 AND   #$F
D249 AA                    TAX
D24A BC EB D1              LDY   COS,X SAVE COS AND SIN
D24D 84 50                 STY   DXL VALS IN DXL AND DY.
D24F 49 0F                 EOR   #$F
D251 AA                    TAX
D252 BC EC D1              LDY   COS+1,X
D255 C8                    INY
D256 84 52                 STY   DY
D258 AC 25 03       DRAW2  LDY   HNDX BYTE INDEX FROM
D25B A2 00                 LDX   #$0 HI-RES BASE ADR.
D25D 8E 2A 03              STX   COLLSN CLEAR COLLISION COUNT.
D260 A1 1A                 LDA   (SHAPEL,X) 1ST SHAPE DEF BYTE.
```

```
D262 85 51        DRAW3  STA   SHAPEX
D264 A2 80               LDX   #$80
D266 86 54               STX   EL  EL,EH FOR FRACTIONAL
D268 86 55               STX   EH  L,R,U,D VECTORS.
D26A AE 27 03            LDX   SCALE SCALE FACTOR.
D26D A5 54        DRAW4  LDA   EL
D26F 38                  SEC   IF FRAC COS OVFL
D270 65 50               ADC   DXL THEN MOVE IN
D272 85 54               STA   EL  SPECIFIED VECTOR
D274 90 04               BCC   DRAW5 DIRECTION.
D276 20 D8 D0            JSR   LRUD1
D279 18                  CLC
D27A A5 55        DRAW5  LDA   EH IF FRAC SIN OVFL
D27C 65 52               ADC   DY THEN MOVE IN
D27E 85 55               STA   EH SPECIFIED VECTOR
D280 90 03               BCC   DRAW6 DIRECTION +90 DEG.
D282 20 D9 D0            JSR   LRUD2
D285 CA           DRAW6  DEX   LOOP ON SCALE
D286 D0 E5               BNE   DRAW4 FACTOR.
D288 A5 51               LDA   SHAPEX
D28A 4A                  LSR   ; NEXT 3-BIT VECTOR
D28B 4A                  LSR   ; OF SHAPE DEF.
D28C 4A                  LSR
D28D D0 D3               BNE   DRAW3 NOT DONE THIS BYTE.
D28F E6 1A               INC   SHAPEL
D291 D0 02               BNE   DRAW7 NEXT BYTE OF
D293 E6 1B               INC   SHAPEH SHAPE DEFINITION.
D295 A1 1A        DRAW7  LDA   (SHAPEL,X)
D297 D0 C9               BNE   DRAW3 DONE IF ZERO.
D299 60                  RTS

                  *  HI-RES GRAPHICS SHAPE EX-OR SUBR
                  *
                  * EX-OR SHAPE INTO SCREEN.
                  *
                  * ROT = 0 TO 3 (QUADRANT ONLY)
                  * SCALE IS USED
                  *
D29A 86 1A        XDRAW  STX   SHAPEL SHAPE DEFINITION
D29C 84 1B               STY   SHAPEH POINTER.
D29E AA           XDRAW1 TAX
D29F 4A                  LSR   ; ROT ($0-$3F)
D2A0 4A                  LSR
D2A1 4A                  LSR   ; QDRNT 0=UP, 1=RT,
D2A2 4A                  LSR   ; 2=DWN, 3=LFT.
D2A3 85 53               STA   QDRNT
D2A5 8A                  TXA
D2A6 29 0F               AND   #$F
D2A8 AA                  TAX
D2A9 BC EB D1            LDY   COS,X SAVE COS AND SIN
D2AC 84 50               STY   DXL VALS IN DXL AND DY,
D2AE 49 0F               EOR   #$F
D2B0 AA                  TAX
D2B1 BC EC D1            LDY   COS+1,X
D2B4 C8                  INY
D2B5 84 52               STY   DY
D2B7 AC 25 03     XDRAW2 LDY   HNDX INDEX FROM HI-RES
D2BA A2 00               LDX   #$0 BASE ADR.
D2BC 8E 2A 03            STX   COLLSN CLEAR COLLISION DETECT
D2BF A1 1A               LDA   (SHAPEL,X) 1ST SHAPE DEF BYTE.
```

```
D2C1 85 51        XDRAW3 STA   SHAPEX
D2C3 A2 80               LDX   #$80
D2C5 86 54               STX   EL  EL,EH FOR FRACTIONAL
D2C7 86 55               STX   EH  L,R,U,D, VECTORS.
D2C9 AE 27 03            LDX   SCALE SCALE FACTOR.
D2CC A5 54        XDRAW4 LDA   EL
D2CE 38                  SEC   IF FRAC COS OVFL
D2CF 65 50               ADC   DXL THEN MOVE IN
D2D1 85 54               STA   EL SPECIFIED VECTOR
D2D3 90 04               BCC   XDRAW5 DIRECTION
D2D5 20 C0 D0            JSR   LRUDX1
D2D8 18                  CLC
D2D9 A5 55        XDRAW5 LDA   EH IF FRAC SIN OVFL
D2DB 65 52               ADC   DY THEN MOVE IN
D2DD 85 55               STA   EH SPECIFIED VECTOR
D2DF 90 03               BCC   XDRAW6 DIRECTION +90 DEG.
D2E1 20 D9 D0            JSR   LRUD2
D2E4 CA           XDRAW6 DEX   LOOP ON SCALE
D2E5 D0 E5               BNE   XDRAW4 FACTOR.
D2E7 A5 51               LDA   SHAPEX
D2E9 4A                  LSR   ; NEXT 3-BIT VECTOR
D2EA 4A                  LSR   ; OF SHAPE DEF.
D2EB 4A                  LSR
D2EC D0 D3               BNE   XDRAW3
D2EE E6 1A               INC   SHAPEL
D2F0 D0 02               BNE   XDRAW7 NEXT BYTE OF
D2F2 E6 1B               INC   SHAPEH SHAPE DEF.
D2F4 A1 1A        XDRAW7 LDA   (SHAPEL,X)
D2F6 D0 C9               BNE   XDRAW3 DONE IF ZERO.
D2F8 60                  RTS

                  *  ENTRY POINTS FROM APPLE-II BASIC
D2F9 20 90 D3     BPOSN  JSR   PCOLR POSN CALL, COLR FROM BASIC
D2FC 8D 24 03            STA   HCOLOR
D2FF 20 AF D3            JSR   GETYO YO FROM BASIC.
D302 48                  PHA
D303 20 9A D3            JSR   GETXO XO FROM BASIC.
D306 68                  PLA
D307 20 2E D0            JSR   HPOSN
D30A AE 23 03            LDX   BXSAV
D30D 60                  RTS
D30E 20 F9 D2     BPLOT  JSR   BPOSN PLOT CALL (BASIC).
D311 4C 7D D0            JMP   HPLOT1
D314 AD 25 03     BLIN1  LDA   HNDX
D317 4A                  LSR   ; SET HCOLOR1 FROM
D318 20 90 D3            JSR   PCOLR BASIC VAR COLR.
D31B 20 75 D0            JSR   HPOSN3
D31E 20 9A D3     BLINE  JSR   GETXO LINE CALL, GET XO FROM BASIC
D321 8A                  TXA
D322 48                  PHA
D323 98                  TYA
D324 AA                  TAX
D325 20 AF D3            JSR   GETYO YO FROM BASIC
D328 A8                  TAY
D329 68                  PLA
D32A 20 64 D1            JSR   HLIN
D32D AE 23 03            LDX   BXSAV
D330 60                  RTS
D331 20 90 D3     BGND   JSR   PCOLR BACKGROUND CALL
D334 4C 10 D0            JMP   BKGND0
```

```
                    *  DRAW ROUTINES
D337 20 F9 D2       BDRAW1 JSR   BPOSN
D33A 20 51 D3       BDRAW  JSR   BDRAWX DRAW CALL FROM BASIC.
D33D 20 3B D2              JSR   DRAW
D340 AE 23 03              LDX   BXSAV
D343 60                    RTS
D344 20 F9 D2       BXDRW1 JSR   BPOSN
D347 20 51 D3       BXDRAW JSR   BDRAWX EX-OR DRAW
D34A 20 9A D2              JSR   XDRAW FROM BASIC.
D34D AE 23 03              LDX   BXSAV
D350 60                    RTS
D351 8E 23 03       BDRAWX STX   BXSAV SAVE FOR BASIC.
D354 A0 32                 LDY   #$32
D356 20 92 D3              JSR   PBYTE SCALE FROM BASIC.
D359 8D 27 03              STA   SCALE
D35C A0 28                 LDY   #$28
D35E 20 92 D3              JSR   PBYTE ROT FROM BASIC.
D361 48                    PHA   SAVE ON STACK.
D362 AD 28 03              LDA   SHAPXL
D365 85 1A                 STA   SHAPEL START OF
D367 AD 29 03              LDA   SHAPXH SHAPE TABLE.
D36A 85 1B                 STA   SHAPEH
D36C A0 20                 LDY   #$20
D36E 20 92 D3              JSR   PBYTE SHAPE FROM BASIC.
D371 F0 39                 BEQ   RERR1
D373 A2 00                 LDX   #$0
D375 C1 1A                 CMP   (SHAPEL,X) > NUM OF SHAPES?
D377 F0 02                 BEQ   BDRWX1
D379 B0 31                 BCS   RERR1 YES, RANGE ERR.
D37B 0A             BDRWX1 ASL
D37C 90 03                 BCC   BDRWX2
D37E E6 1B                 INC   SHAPEH
D380 18                    CLC
D381 A8             BDRWX2 TAY   SHAPE NO. * 2.
D382 B1 1A                 LDA   (SHAPEL),Y
D384 65 1A                 ADC   SHAPEL
D386 AA                    TAX   ADD 2-BYTE INDEX
D387 C8                    INY   TO SHAPE TABLE
D388 B1 1A                 LDA   (SHAPEL),Y START ADR
D38A 6D 29 03              ADC   SHAPXH (X LOW, Y HI).
D38D A8                    TAY
D38E 68                    PLA   ROT FROM STACK.
D38F 60                    RTS

                    *  BASIC PARAM FETCH SUBR'S
D390 A0 16          PCOLR  LDY   #$16
D392 B1 4A          PBYTE  LDA   (LOMEML),Y
D394 D0 16                 BNE   RERR1 GET BASIC PARAM.
D396 88                    DEY   (ERR IF >255)
D397 B1 4A                 LDA   (LOMEML),Y
D399 60             RTSB   RTS
D39A 8E 23 03       GETX0  STX   BXSAV SAVE FOR BASIC.
D39D A0 05                 LDY   #$5
D39F B1 4A                 LDA   (LOMEML),Y X0 LOW-ORDER BYTE.
D3A1 AA                    TAX
D3A2 C8                    INY
D3A3 B1 4A                 LDA   (LOMEML),Y HI-ORDER BYTE.
D3A5 A8                    TAY
D3A6 E0 18                 CPX   #$18
D3A8 E9 01                 SBC   #$1 RANGE ERR IF >279
D3AA 90 ED                 BCC   RTSB
D3AC 4C 68 EE       RERR1  JMP   RNGERR
D3AF A0 0D          GETY0  LDY   #$D OFFSET TO Y0 FROM LOMEM
D3B1 20 92 D3              JSR   PBYTE GET BASIC PARAM Y0.
D3B4 C9 C0                 CMP   #$C0 (ERR IF >191)
D3B6 B0 F4                 BCS   RERR1
D3B8 60                    RTS
```

```
                    *  SHAPE TAPE LOAD SUBROUTINE
D3B9 8E 23 03       SHLOAD STX    BXSAV SAVE FOR BASIC.
D3BC 20 1E F1              JSR    ACADR READ 2-BYTE LENGTH INTO
D3BF 20 FD FE              JSR    READ BASIC ACC
D3C2 A9 00                 LDA    #$00 ;START OF SHAPE TABLE IS $0800
D3C4 85 3C                 STA    A1L
D3C6 8D 28 03              STA    SHAPXL
D3C9 18                    CLC
D3CA 65 CE                 ADC    ACL
D3CC A8                    TAY
D3CD A9 08                 LDA    #$08 ;HIGH BYTE OF SHAPE TABLE POINTER.
D3CF 85 3D                 STA    A1H
D3D1 8D 29 03              STA    SHAPXH
D3D4 65 CF                 ADC    ACH
D3D6 B0 25                 BCS    MFULL1 NOT ENOUGH MEMORY.
D3D8 C4 CA                 CPY    PPL
D3DA 48                    PHA
D3DB E5 CB                 SBC    PPH
D3DD 68                    PLA
D3DE B0 1D                 BCS    MFULL1
D3E0 84 3E                 STY    A2L
D3E2 85 3F                 STA    A2H
D3E4 C8                    INY
D3E5 D0 02                 BNE    SHLOD1
D3E7 69 01                 ADC    #$1
D3E9 84 4A          SHLOD1 STY    LOMEML
D3EB 85 4B                 STA    LOMEMH
D3ED 84 CC                 STY    PVL
D3EF 85 CD                 STA    PVH
D3F1 20 FA FC              JSR    RD2BIT
D3F4 A9 03                 LDA    #$3 .5 SECOND HEADER.
D3F6 20 02 FF              JSR    READX1
D3F9 AE 23 03              LDX    BXSAV
D3FC 60                    RTS
D3FD 4C 6B E3       MFULL1 JMP    MEMFUL

--- END ASSEMBLY ---

TOTAL ERRORS: 00
```

```
*************************************************
*                                               *
*  APPLE-II BASIC RENUMBER / APPEND SUBROUTINES  *
*                                               *
*                VERSION  TWO                   *
*                  RENUMBER                     *
*                 >CLR                          *
*                 >START=                       *
*                 >STEP=                        *
*                 >CALL -10531                  *
*                                               *
*                  OPTIONAL                     *
*                 >FROM=                        *
*                 >TO=                          *
*                 >CALL -10521                  *
*                                               *
*               USE RENX ENTRY                  *
*              FOR RENUMBER ALL                 *
*                                               *
*          WOZ     APRIL 12, 1978               *
*             APPLE COMPUTER INC.               *
*************************************************

*
*
*          6502 EQUATES
*
ROL      EQU   $0            LOW-ORDER SW16 R0 BYTE.
ROH      EQU   $1             HI-ORDER.
ONE      EQU   $01
R11L     EQU   $16           LOW-ORDER SW16 R11 BYTE.
R11H     EQU   $17            HI-ORDER.
HIMEM    EQU   $4C           BASIC HIMEM POINTER.
PPL      EQU   $CA           BASIC PROG POINTER.
PVL      EQU   $CC           BASIC VAR POINTER.
MEMFULL EQU    $E36B          BASIC MEM FULL ERROR.
PRDEC    EQU   $E51B         BASIC DECIMAL PRINT SUBR.
RANGERR EQU    $EE68          BASIC RANGE ERROR.
LOAD     EQU   $F0DF         BASIC LOAD SUBR.
SW16     EQU   $F689         SWFET 16 ENTRY.
CROUT    EQU   $FD8E         CAR RET SUBR.
COUT     EQU   $FDED         CHAR OUT SUBR.

*
*          SWEET 16 EQUATES
*
ACC      EQU   $0            SWEET 16 ACCUMULATOR.
NEWLOW EQU     $1            NEW INITIAL LNO.
NEWINCR EQU     $2            NEW LNO INCR.
LNLOW    EQU   $3            LOW LNO OF RENUM RANGE.
LNHI     EQU   $4            HI LNO OF RENUM RANGE.
TBLSTRT EQU     $5            LNO TABLE START.
TBLNDX1 EQU     $6            PASS 1 LNO TBL INDEX.
TBLIM    EQU   $7            LNO TABLE LIMIT.
SCR8     EQU   $8            SCRATCH REG.
HMEM     EQU   $8            HIMEM (END OF PRGM).
SCR9     EQU   $9            SCRATCH REG.
PRGNDX EQU     $9            PASS 1 PROG INDEX.
PRGNDX1 EQU     $A            ALSO PROG INDEX.
NEWLN    EQU   $B            NEXT "NEW LNO".
NEWLN1 EQU     $C            PRIOR "NEW LNO" ASSIGN.
TBLND    EQU   $6            PASS 2 LNO TABLE END.
PRGNDX2 EQU     $7            PASS 2 PROG INDEX.
CHR0     EQU   $9            ASCII "0".
CHRA     EQU   $A            ASCII "A".
```

```
          MODE     EQU     $C           CONST/LNO MODE.
          TBLNDX2  EQU     $B            LNO TBL IDX FOR UPDATE.
          OLDLN    EQU     $D           OLD LNO FOR UPDATE.
          STRCON   EQU     $B           BASIC STR CON TOKEN.
          REM      EQU     $C           BASIC REM TOKEN.
          R13      EQU     $D           SWEET 16 REG 13 (CPR REG).
          THEN     EQU     $D           BASIC THEN TOKEN.
          LIST     EQU     $D           BASIC LIST TOKEN.
          DEL      EQU     $D
          SCRC     EQU     $C           SCRATCH REG FOR APPEND.

          *
          *            APPLE-11 BASIC RENUMBER SUBROUTINE - PASS 1
                   ORG     $D400
                   OBJ     $A400
D400 20 89 F6  RENX     JSR     SW16         OPTIONAL RANGE ENTRY.
D403 B0                 SUB     ACC
D404 33                 ST      LNLOW        SET LNLOW=0, LNHI=0.
D405 34                 ST      LNHI
D406 F4                 DCR     LNHI
D407 00                 RTN
D408 20 89 F6  RENUM    JSR     SW16
D40B 18 4C 00           SET     HMEM,HIMEM
D40E 68                 LDD     @HMEM
D40F 38                 ST      HMEM
D410 19 CE 00  RNUM3    SET     SCR9,PVL+2
D413 C9                 POPD    @SCR9        BASIC VAR PNT TO
D414 35                 ST      TBLSTRT       TBLSTRT AND TBLNDX1.
D415 36                 ST      TBLNDX1
D416 21                 LD      NEWLOW       COPY NEWLOW (INITIAL)
D417 3B                 ST      NEWLN         TO NEWLN.
D418 3C                 ST      NEWLN1
D419 C9                 POPD    @SCR9        BASIC PROG PNTR
D41A 37                 ST      TBLIM         TO TBLIM AND PRGNDX.
D41B 39                 ST      PRGNDX
D41C 29        PASS1    LD      PRGNDX
D41D D8                 CPR     HMEM         IF PRGNDX >= HMEM
D41E 03 46              BC      PASS2         THEN DONE PASS 1.
D420 3A                 ST      PRGNDX1
D421 26                 LD      TBLNDX1
D422 E0                 INR     ACC          IF < TWO BYTES AVAIL IN
D423 D7                 CPR     TBLIM         LNO TABLE THEN RETURN
D424 03 38              BC      MERR          WITH "MEM FULL" MESSAGE.
D426 4A                 LD      @PRGNDX1
D427 A9                 ADD     PRGNDX       ADD LENTH BYTE TO PROG INDEX.
D428 39                 ST      PRGNDX
D429 6A                 LDD     @PRGNDX1     LINE NUMBER.
D42A D3                 CPR     LNLOW        IF < LNLOW THEN GOTO P1B.
D42B 02 2A              BNC     P1B
D42D D4                 CPR     LNHI         IF > LNHI THEN GOTO P1C.
D42E 02 02              BNC     P1A
D430 07 30              BNZ     P1C
D432 76        P1A      STD     @TBLNDX1     ADD TO LNO TABLE.
D433 00                 RTN
D434 A5 01              LDA     R0H          **** 6502 CODE ****
D436 A6 00              LDX     R0L
D438 20 1B E5           JSR     PRDEC        PRINT OLD LNO "->" NEW LNO
D43B A9 AD              LDA     #$AD          (R0,R11) IN DECIMAL.
D43D 20 ED FD           JSR     COUT
D440 A9 BE              LDA     #$BE
D442 20 ED FD           JSR     COUT
D445 A5 17              LDA     R11H
D447 A6 16              LDX     R11L
D449 20 1B E5           JSR     PRDEC
D44C 20 8E FD           JSR     CROUT
               *
D44F 20 8C F6           JSR     SW16+3       **** END 6502 CODE ****
```

```
D452 2B                     LD     NEWLN
D453 3C                     ST     NEWLN1        COPY NEWLN TO NEWLN1 AND INCR
D454 A2                     ADD    NEWINCR        NEWLN BY NEWINCR.
D455 3B                     ST     NEWLN
D456 0D                     HEX    0D            'NUL' (WILL SKIP NEXT INSTRUCTION)
D457 D1             P1B     CPR    NEWLOW        IF LOW LNO < NEW LOW THEN RANGE ERR.
D458 02 C2                  BNC    PASS1
D45A 00             RERR    RTN    PRINT "RANGE ERR" MESSAGE AND RETURN.
D45B 4C 68 EE               JMP    RANGERR
D45E 00             MERR    RTN    PRINT "MEM FULL" MESSAGE AND RETURN.
D45F 4C 6B E3               JMP    MEMFULL
D462 EC             P1C     INR    NEWLN1        IF HI LNO <= MOST RECENT NEWLN THEN
D463 DC                     CPR    NEWLN1         RANGE ERROR.
D464 02 F4                  BNC    RERR

                    *
                    *          APPLE II BASIC RENUMBER / APPEND SUBROUTINE - PASS 2
                    *
D466 19 B0 00       PASS2   SET    CHR0,$00B0      ASCII "0".
D469 1A C0 00               SET    CHRA,$00C0      ASCII "A".
D46C 27             P2A     LD     PRGNDX2
D46D D8                     CPR    HMEM          IF PROG INDEX = HIMEM THEN DONE PASS 2.
D46E 03 63                  BC     DONE
D470 E7                     INR    PRGNDX2       SKIP LENTH BYTE.
D471 67                     LDD    @PRGNDX2      LINE NUMBER.
D472 3D             UPDATE  ST     OLDLN         SAVE OLD LNO.
D473 25                     LD     TBLSTRT
D474 3B                     ST     TBLNDX2       INIT LNO TABLE INDEX.
D475 21                     LD     NEWLOW        INIT NEWLN TO NEWLOW.
D476 1C                     HEX    1C            (WILL SKIP NEXT INSTR)
D477 2C             UD2     LD     NEWLN1
D478 A2                     ADD    NEWINCR       ADD INCR TO NEWLN1.
D479 3C                     ST     NEWLN1
D47A 2B                     LD     TBLNDX2       IF LNO TBL IDX = TBLND THEN DONE
D47B B6                     SUB    TBLND          SCANNING LNO TABLE
D47C 03 07                  BC     UD3
D47E 6B                     LDD    @TBLNDX2      NEXT LNO FROM TABLE.
D47F BD                     SUB    OLDLN         LOOP TO UD2 IF NOT SAME AS OLDLN.
D480 07 F5                  BNZ    UD2
D482 C7                     POPD   @PRGNDX2      REPLACE OLD LNO WITH CORRESPONDING
D483 2C                     LD     NEWLN1         NEW LINE.
D484 77                     STD    @PRGNDX2
D485 1B 28 00       UD3     SET    STRCON,$0028   STR CON TOKEN.
D488 1C                     HEX    1C            (SKIPS NEXT TWO INSTRUCTIONS)
D489 67             GOTCON  LDD    @PRGNDX2
D48A FC                     DCR    MODE          IF MODE = 0 THEN UPDATE LNO REF.
D48B 08 E5                  BM1    UPDATE
D48D 47             ITEM    LD     @PRGNDX2      BASIC TOKEN.
D48E D9                     CPR    CHR0
D48F 02 09                  BNC    CHKTOK        CHECK TOKEN FOR SPECIAL.
D491 DA                     CPR    CHRA          IF >= "0" AND < "A" THEN SKIP CONST
D492 02 F5                  BNC    GOTCON         OR UPDATE.
D494 F7             SKPASC  DCR    PRGNDX2
D495 67                     LDD    @PRGNDX2      SKIP ALL NEG. BYTES OF STR CON, REM,
D496 05 FC                  BM     SKPASC         OR NAME.
D498 F7                     DCR    PRGNDX2
D499 47                     LD     @PRGNDX2
```

```
D49A DB          CHKTOK CPR    STRCON       STR CON TOKEN?
D49B 06 F7              BZ     SKPASC        YES, SKIP SUBSEQUENT BYTES.
D49D 1C 5D 00           SET    REM,$005D
D4A0 DC                 CPR    REM          REM TOKEN?
D4A1 06 F1              BZ     SKPASC        YES, SKIP SUBSEQUENT LINE.
D4A3 08 13              BM1    CONTST       GOSUB, LOOK FOR LINE NUMBER.
D4A5 FD                 DCR    R13
D4A6 FD                 DCR    R13          (TOKEN $5F IS GOTO)
D4A7 06 0F              BZ     CONTST
D4A9 1D 24 00           SET    THEN,$0024
D4AC DD                 CPR    THEN
D4AD 06 09              BZ     CONTST       'THEN' LNO, LOOK FOR LNO.
D4AF F0                 DCR    ACC
D4B0 06 BA              BZ     P2A          EOL (TOKEN 01)?
D4B2 1D 74 00           SET    LIST,$0074
D4B5 BD                 SUB    LIST         SET MODEIF  LIST OR LIST COMMA.
D4B6 09 01              BNM1   CONTS2        (TOKENS $74, $75)
D4B8 B0          CONTST SUB    ACC          CLEAR MODE FOR LNO
D4B9 3C          CONTS2 ST     MODE          UPDATE CHECK.
D4BA 01 D1              BR     ITEM
                 *
                 *
                 *          APPLE II BASIC APPEND SUBROUTINE
                 *
D4BC 20 89 F6    APPEND JSR    SW16
D4BF 1C 4E 00           SET    SCRC,HIMEM+2
D4C2 CC                 POPD   @SCRC         SAVE HIMEM.
D4C3 38                 ST     HMEM
D4C4 19 CA 00           SET    SCR9,PPL
D4C7 69                 LDD    @SCR9
D4C8 7C                 STD    @SCRC         SET HIMEM TO PRESERVE PROGRAM.
D4C9 00                 RTN
D4CA 20 DF F0           JSR    LOAD          LOAD FROM TAPE.
D4CD 20 89 F6           JSR    SW16
D4D0 CC                 POPD   @SCRC         RESTORE HIMEM TO SHOW BOTH PROGRAMS.
D4D1 28                 LD     HMEM             (OLD AND NEW)
D4D2 7C                 STD    @SCRC
D4D3 00          DONE   RTN    RETURN.
D4D4 60                 RTS

---- END ASSEMBLY ----

TOTAL ERRORS: 00
```

:ASM

```
****************************
*                          *
*     6502 RELOCATION      *
*        SUBROUTINE        *
*                          *
*   1. DEFINE BLOCKS       *
*      *A4<A1.A2 ^Y        *
*      (^Y IS CTRL-Y)      *
*                          *
*   2. FIRST SEGMENT       *
*      *A4<A1.A2 ^Y        *
*         (IF CODE)        *
*                          *
*      *A4<A1.A2M          *
*         (IF MOVE)        *
*                          *
*   3. SUBSEQUENT SEGMENTS *
*      *.A2 ^Y OR *.A2M    *
*                          *
*     WOZ  11-10-77        *
*   APPLE COMPUTER INC.    *
*                          *
****************************

*
*    RELOCATION SUBROUTINE EQUATES
*
R1L    EQU    $02 SWEET 16 REG 1.
INST   EQU    $0B 3-BYTE INST FIELD.
LENGTH EQU    $2F LENGTH CODE
YSAV   EQU    $34 CMND BUF POINTER
A1L    EQU    $3C APPLE-II MON PARAM AREA.
A4L    EQU    $42 APPLE-II MON PARAM REG 4
IN     EQU    $0200
SW16   EQU    $F689 ;SWEET 16 ENTRY
INSDS2 EQU    $F88E ;DISASSEMBLER ENTRY
NXTA4  EQU    $FCB4 POINTER INCR SUBR
FRMBEG EQU    $01 SOURCE BLOCK BEGIN
FRMEND EQU    $02 SOURCE BLOCK END
TOBEG  EQU    $04 DEST BLOCK BEGIN
ADR    EQU    $06 ADR PART OF INST.
```

```
                   *
                   *   6502 RELOCATION SUBROUTINE
                   *
                           ORG    $D4DC
                           OBJ    $A4DC
D4DC A4 34         RELOC   LDY    YSAV CMND BUF POINTER
D4DE B9 00 02              LDA    IN,Y NEXT CMD CHAR
D4E1 C9 AA                 CMP    #$AA '*'?
D4E3 D0 0C                 BNE    RELOC2   NO, RELOC CODE SEG.
D4E5 E6 34                 INC    YSAV ADVANCE POINTER.
D4E7 A2 07                 LDX    #$07
D4E9 B5 3C         INIT    LDA    A1L,X MOVE BLOCK PARAMS
D4EB 95 02                 STA    R1L,X   FROM APPLE-II MON
D4ED CA                    DEX    AREA TO SW16 AREA
D4EE 10 F9                 BPL    INIT   R1=SOURCE BEG, R2=
D4F0 60                    RTS    SOURCE END, R4=DEST BEG.
D4F1 A0 02         RELOC2  LDY    #$02
D4F3 B1 3C         GETINS  LDA    (A1L),Y COPY 3 BYTES TO
D4F5 99 0B 00              STA    INST,Y   SW16 AREA
D4F8 88                    DEY
D4F9 10 F8                 BPL    GETINS
D4FB 20 8E F8              JSR    INSDS2 CALCULATE LENGTH OF
D4FE A6 2F                 LDX    LENGTH   INST FROM OPCODE.
D500 CA                    DEX    0=1 BYTE, 1=2 BYTES,
D501 D0 0C                 BNE    XLATE 2=3 BYTES.
D503 A5 0B                 LDA    INST
D505 29 0D                 AND    #$0D WEED OUT NON-ZERO-PAGE
D507 F0 14                 BEQ    STINST   2 BYTE INSTS (IMM).
D509 29 08                 AND    #$08   IF ZERO PAGE ADR
D50B D0 10                 BNE    STINST  THEN CLEAR HIGH BYTE
D50D 85 0D                 STA    INST+2
D50F 20 89 F6      XLATE   JSR    SW16 IF ADR OF ZERO PAGE
D512 22                    LD     FRMEND   OR ABS IS IN SOURCE
D513 D6                    CPR    ADR   (FRM) BLOCK THEN
D514 02 06                 BNC    SW16RT   SUBSTITUTE
D516 26                    LD     ADR    ADR-SOURCE BEG+DEST BEG
D517 B1                    SUB    FRMBEG
D518 02 02                 BNC    SW16RT
D51A A4                    ADD    TOBEG
D51B 36                    ST     ADR
D51C 00            SW16RT  RTN
D51D A2 00         STINST  LDX    #$00
D51F B5 0B         STINS2  LDA    INST,X
D521 91 42                 STA    (A4L),Y COPY LENGTH BYTES
D523 E8                    INX    OF INST FROM SW16 AREA TO
D524 20 B4 FC              JSR    NXTA4
D527 C6 2F                 DEC    LENGTH   DEST SEGMENT. UPDATE
D529 10 F4                 BPL    STINS2   SOURCE,DEST SEGMENT
D52B 90 C4                 BCC    RELOC2   POINTERS.  LOOP IF NOT
D52D 60                    RTS    BEYOND SOURCE SEG END.

--- END ASSEMBLY ---

TOTAL ERRORS: 00
```

```
                 ******************************
                 *                            *
                 *        TAPE VERIFY         *
                 *                            *
                 *          JAN 78            *
                 *          BY WOZ            *
                 *                            *
                 *                            *
                 ******************************

                 *
                 *      TAPE VERIFY EQUATES
                 *
                 CHKSUM EQU   $2E
                 A1     EQU   $3C
                 HIMEM  EQU   $4C ;BASIC HIMEM POINTER
                 PP     EQU   $CA ;BASIC BEGIN OF PROGRAM
                 PRLEN  EQU   $CE ;BASIC PROGRAM LENGTH
                 XSAVE  EQU   $D8 ;PRESERVE X-REG FOR BASIC
                 HDRSET EQU   $F11E ;SETS TAPE POINTERS TO $CE.CF
                 PRGSET EQU   $F12C ;SETS TAPE POINTERS FOR PROGRAM
                 NXTA1  EQU   $FCBA ;INCREMENTS (A1) AND COMPARES TO (A2)
                 HEADR  EQU   $FCC9
                 RDBYTE EQU   $FCEC
                 RD2BIT EQU   $FCFA
                 RDBIT  EQU   $FCFD
                 PRA1   EQU   $FD92 ;PRINT (A1)-
                 PRBYTE EQU   $FDDA
                 COUT   EQU   $FDED
                 FINISH EQU   $FF26 ;CHECK CHECKSUM, RING BELL
                 PRERR  EQU   $FF2D

                 *
                 *       TAPE VERIFY ROUTINE
                 *
                        ORG   $D535
                        OBJ   $A535
D535 86 D8       VFYBSC STX   XSAVE ;PRESERVE X-REG FOR BASIC
D537 38                 SEC
D538 A2 FF              LDX   #$FF
D53A A5 4D       GETLEN LDA   HIMEM+1 ;CALCULATE PROGRAM LENGTH
D53C F5 CB              SBC   PP+1,X ;INTO PRLEN
D53E 95 CF              STA   PRLEN+1,X
D540 E8                 INX
D541 F0 F7              BEQ   GETLEN
D543 20 1E F1           JSR   HDRSET ;SET UP POINTERS
D546 20 54 D5           JSR   TAPEVFY ;DO A VERIFY ON HEADER
D549 A2 01              LDX   #$01 ;PREPARE FOR PRGSET
D54B 20 2C F1           JSR   PRGSET ;SET POINTERS FOR PROGRAM VERIFY
D54E 20 54 D5           JSR   TAPEVFY
D551 A6 D8              LDX   XSAVE ;RESTORE X-REG
D553 60                 RTS
```

```
                  *
                  * TAPE VERIFY RAM IMAGE (A1.A2)
                  *
D554 20 FA FC     TAPEVFY JSR   RD2BIT
D557 A9 16                LDA   #$16
D559 20 C9 FC             JSR   HEADR ;SYNCHRONIZE ON HEADER
D55C 85 2E                STA   CHKSUM ;INITIALIZE CHKSUM
D55E 20 FA FC             JSR   RD2BIT
D561 A0 24        VRFY2   LDY   #$24
D563 20 FD FC             JSR   RDBIT
D566 B0 F9                BCS   VRFY2 ;CARRY SET IF READ A '1' BIT
D568 20 FD FC             JSR   RDBIT
D56B A0 3B                LDY   #$3B
D56D 20 EC FC     VRFY3   JSR   RDBYTE ;READ A BYTE
D570 F0 0E                BEQ   EXTDEL ;ALWAYS TAKEN
D572 45 2E        VFYLOOP EOR   CHKSUM ;UPDATE CHECKSUM
D574 85 2E                STA   CHKSUM
D576 20 BA FC             JSR   NXTA1 ;INCREMENT A1, SET CARRY IF A1>A2
D579 A0 34                LDY   #$34 ;ONE LESS THAN USED IN READ FOR EXTRA 12
D57B 90 F0                BCC   VRFY3 ;LOOP UNTIL A1>A2
D57D 4C 26 FF             JMP   FINISH ;VERIFY CHECKSUM&RING BELL
D580 EA           EXTDEL  NOP   ;EXTRA DELAY TO EQUALIZE TIMING
D581 EA                   NOP   ;     (+12 USEC)
D582 EA                   NOP
D583 C1 3C                CMP   (A1,X) ;BYTE THE SAME?
D585 F0 EB                BEQ   VFYLOOP ;IT MATCHES, LOOP BACK
D587 48                   PHA   ;SAVE WRONG BYTE FROM TAPE
D588 20 2D FF             JSR   PRERR ;PRINT "ERR"
D58B 20 92 FD             JSR   PRA1 ;OUTPUT (A1)"-"
D58E B1 3C                LDA   (A1),Y
D590 20 DA FD             JSR   PRBYTE ;OUTPUT CONTENTS OF A1
D593 A9 A0                LDA   #$A0 ;PRINT A BLANK
D595 20 ED FD             JSR   COUT
D598 A9 A8                LDA   #$A8 ;  '('
D59A 20 ED FD             JSR   COUT
D59D 68                   PLA   ;OUTPUT BAD BYTE FROM TAPE
D59E 20 DA FD             JSR   PRBYTE
D5A1 A9 A9                LDA   #$A9 ;   ')'
D5A3 20 ED FD             JSR   COUT
D5A6 A9 8D                LDA   #$8D ;CARRIAGE RETURN, AND RETURN TO CALLER
D5A8 4C ED FD             JMP   COUT

--- END ASSEMBLY ---

TOTAL ERRORS: 00
```

:ASM

```
***************************
*                         *
*        RAMTEST:         *
*                         *
*         BY WOZ          *
*          6/77           *
*                         *
*   COPYRIGHT 1978 BY:    *
*   APPLE COMPUTER INC    *
*                         *
***************************

*
*        EQUATES:
*
DATA     EQU    $0 TEST DATA $00 OR $FF
NDATA    EQU    $1 INVERSE TEST DATA.
TESTD    EQU    $2 GALLOP DATA.
R3L      EQU    $6 AUX ADR POINTER.
R3H      EQU    $7
R4L      EQU    $8 AUX ADR POINTER.
R4H      EQU    $9
R5L      EQU    $A AUX ADR POINTER.
R5H      EQU    $B
R6L      EQU    $C GALLOP BIT MASK.
R6H      EQU    $D ($0001 TO 2^N)
YSAV     EQU    $34 MONITOR SCAN INDEX.
A1H      EQU    $3D BEGIN TEST BLOCK ADR.
A2L      EQU    $3E LEN (PAGES) FROM MON.
SETCTLY EQU      $D5B0 ;SET UP CNTRL-Y LOCATION
PRBYTE EQU      $FDDA BYTE PRINT SUBR.
COUT     EQU    $FDED CHAR OUT SUBR.
PRERR    EQU    $FF2D PRINTS 'ERR-BELL'
BELL     EQU    $FF3A
```

```
               *
               *      RAMTEST:
               *
                             ORG   $D5BC
                             OBJ   $A5BC
D5BC A9 C3           SETUP   LDA   #$C3 ;SET UP CNTRL-Y LOCATION
D5BE A0 D5                   LDY   #$D5
D5C0 4C B0 D5                JMP   SETCTLY
D5C3 A9 00           RAMTST  LDA   #$0 TEST FOR $00,
D5C5 20 D0 D5                JSR   TEST
D5C8 A9 FF                   LDA   #$FF THEN $FF.
D5CA 20 D0 D5                JSR   TEST
D5CD 4C 3A FF                JMP   BELL
D5D0 85 00           TEST    STA   DATA
D5D2 49 FF                   EOR   #$FF
D5D4 85 01                   STA   NDATA
D5D6 A5 3D                   LDA   A1H
D5D8 85 07                   STA   R3H INIT (R3L,R3H),
D5DA 85 09                   STA   R4H   (R4L,R4H), (R5L,R5H)
D5DC 85 0B                   STA   R5H   TO TEST BLOCK BEGIN
D5DE A0 00                   LDY   #$0   ADDRESS.
D5E0 84 06                   STY   R3L
D5E2 84 08                   STY   R4L
D5E4 84 0A                   STY   R5L
D5E6 A6 3E                   LDX   A2L LENGTH (PAGES).
D5E8 A5 00                   LDA   DATA
D5EA 91 08           TEST01  STA   (R4L),Y SET ENTIRE TEST
D5EC C8                      INY   BLOCK TO DATA.
D5ED D0 FB                   BNE   TEST01
D5EF E6 09                   INC   R4H
D5F1 CA                      DEX
D5F2 D0 F6                   BNE   TEST01
D5F4 A6 3E                   LDX   A2L
D5F6 B1 06           TEST02  LDA   (R3L),Y VERIFY ENTIRE
D5F8 C5 00                   CMP   DATA   TEST BLOCK.
D5FA F0 13                   BEQ   TEST03
D5FC 48                      PHA   PRESERVE BAD DATA.
D5FD A5 07                   LDA   R3H
D5FF 20 DA FD                JSR   PRBYTE PRINT ADDRESS,
D602 98                      TYA
D603 20 8A D6                JSR   PRBYSP
D606 A5 00                   LDA   DATA THEN EXPECTED DATA,
D608 20 8A D6                JSR   PRBYSP
D60B 68                      PLA   THEN BAD DATA,
D60C 20 7F D6                JSR   PRBYCR   THEN 'ERR-BELL'.
D60F C8              TEST03  INY
D610 D0 E4                   BNE   TEST02
D612 E6 07                   INC   R3H
D614 CA                      DEX
D615 D0 DF                   BNE   TEST02
D617 A6 3E                   LDX   A2L LENGTH.
D619 A5 01           TEST04  LDA   NDATA
D61B 91 0A                   STA   (R5L),Y SET TEST CELL TO
D61D 84 0D                   STY   R6H   NDATA AND R6
D61F 84 0C                   STY   R6L   (GALLOP BIT MASK)
D621 E6 0C                   INC   R6L   TO $0001.
D623 A5 01           TEST05  LDA   NDATA
D625 20 45 D6                JSR   TEST6 GALLOP WITH NDATA.
D628 A5 00                   LDA   DATA
D62A 20 45 D6                JSR   TEST6 THEN WITH DATA.
D62D 06 0C                   ASL   R6L
D62F 26 0D                   ROL   R6H SHIFT GALLOP BIT
D631 A5 0D                   LDA   R6H   MASK FOR NEXT
```

```
D633 C5 3E               CMP   A2L   NEIGHBOR.  DONE
D635 90 EC               BCC   TEST05    IF > LENGTH.
D637 A5 00               LDA   DATA
D639 91 0A               STA   (R5L),Y RESTORE TEST CELL.
D63B E6 0A               INC   R5L
D63D D0 DA               BNE   TEST04
D63F E6 0B               INC   R5H INCR TEST CELL
D641 CA                  DEX   POINTER AND DECR
D642 D0 D5               BNE   TEST04   LENGTH COUNT.
D644 60          RTS1    RTS
D645 85 02       TEST6   STA   TESTD SAVE GALLOP DATA.
D647 A5 0A               LDA   R5L
D649 45 0C               EOR   R6L SET R4 TO R5
D64B 85 08               STA   R4L   EX-OR R6
D64D A5 0B               LDA   R5H   FOR NEIGHBOR
D64F 45 0D               EOR   R6H   ADDRESS (1 BIT
D651 85 09               STA   R4H   DIFFERENCE).
D653 A5 02               LDA   TESTD
D655 91 08               STA   (R4L),Y GALLOP TEST DATA.
D657 B1 0A               LDA   (R5L),Y CHECK TEST CELL
D659 C5 01               CMP   NDATA    FOR CHANGE.
D65B F0 E7               BEQ   RTS1 (OK).
D65D 48                  PHA   PRESERVE FAIL DATA.
D65E A5 0B               LDA   R5H
D660 20 DA FD            JSR   PRBYTE PRINT TEST CELL
D663 A5 0A               LDA   R5L   ADDRESS.
D665 20 8A D6            JSR   PRBYSP
D668 A5 01               LDA   NDATA
D66A 91 0A               STA   (R5L),Y (REPLACE CORRECT DATA)
D66C 20 8A D6            JSR   PRBYSP    THEN TEST DATA BYTE.
D66F 68                  PLA
D670 20 8A D6            JSR   PRBYSP    THEN FAIL DATA.
D673 A5 09               LDA   R4H
D675 20 DA FD            JSR   PRBYTE
D678 A5 08               LDA   R4L   THEN NEIGHBOR ADR.
D67A 20 8A D6            JSR   PRBYSP
D67D A5 02               LDA   TESTD   THEN GALLOP DATA.
D67F 20 8A D6    PRBYCR  JSR   PRBYSP OUTPUT BYTE, SPACE.
D682 20 2D FF            JSR   PRERR    THEN 'ERR-BELL'.
D685 A9 8D               LDA   #$8D ASCII CAR. RETURN.
D687 4C ED FD            JMP   COUT
D68A 20 DA FD    PRBYSP  JSR   PRBYTE
D68D A9 A0               LDA   #$A0 OUTPUT BYTE, THEN
D68F 4C ED FD            JMP   COUT    SPACE.
                         ORG   $3F8
03F8 4C C3 D5    USRLOC  JMP   RAMTST ENTRY FROM MON (CTRL-Y)

--- END ASSEMBLY ---

TOTAL ERRORS: 00
```

```
                  ********************************
                  *
                  * MUSIC SUBROUTINE
                  *
                  * GARY J. SHANNON
                  *
                  ********************************
                           ORG   $D717
                  *
                  * ZERO PAGE WORK AREAS
                  * PARAMETER PASSING AREAS
                  *
                  DOWNTIME EQU   $0
                  UPTIME EQU    $1
                  LENGTH EQU    $2
                  VOICE   EQU   $2FD
                  LONG    EQU   $2FE
                  NOTE    EQU   $2FF
                  SPEAKER EQU    $C030
D717 4C 4E D7     ENTRY   JMP   LOOKUP
                  *
                  * PLAY ONE NOTE
                  *
                  * DUTY CYCLE DATA IN 'UPTIME' AND
                  * 'DOWNTIME', DURATION IN 'LENGTH'
                  *
                  *
                  * CYCLE IS DIVIDED INTO 'UP' HALF
                  * AND 'DOWN' HALF
                  *
D71A A4 01        PLAY    LDY   UPTIME ; GET POSITIVE PULSE WIDTH
D71C AD 30 C0             LDA   SPEAKER ; TOGGLE SPEAKER
D71F E6 02        PLAY2   INC   LENGTH ; DURATION
D721 D0 05                BNE   PATH1 ; NOT EXPIRED
D723 E6 03                INC   LENGTH+1
D725 D0 05                BNE   PATH2
D727 60                   RTS   ; DURATION EXPIRED
D728 EA           PATH1   NOP   ; DUMMY
D729 4C 2C D7             JMP   PATH2 ; TIME ADJUSTMENTS
D72C 88           PATH2   DEY   ; DECREMENT WIDTH
D72D F0 05                BEQ   DOWN  ; WIDTH EXPIRED
D72F 4C 32 D7             JMP   PATH3 ; IF NOT, USE UP
                  *
                  * DOWN HALF OF CYCLE
                  *
D732 D0 EB        PATH3   BNE   PLAY2 ; SAME # CYCLES
D734 A4 00        DOWN    LDY   DOWNTIME ; GET NEGATIVE PULSE WIDTH
D736 AD 30 C0             LDA   SPEAKER ; TOGGLE SPEAKER
D739 E6 02        PLAY3   INC   LENGTH ; DURATION
D73B D0 05                BNE   PATH4 ; NOT EXPIRED
D73D E6 03                INC   LENGTH+1
D73F D0 05                BNE   PATH5
D741 60                   RTS   ; DURATION EXPIRED
D742 EA           PATH4   NOP   ; DUMMY
D743 4C 46 D7             JMP   PATH5 ; TIME ADJUSTMENTS
D746 88           PATH5   DEY   ; DECREMENT WIDTH
D747 F0 D1                BEQ   PLAY ; BACK TO UP-SIDE
D749 4C 4C D7             JMP   PATH6 ; USE UP SOME CYCLES
D74C D0 EB        PATH6   BNE   PLAY3 ; REPEAT
```

```
                *
                * NOTE TABLE LOOKUP SUBROUTINE
                *
                * GIVEN NOTE NUMBER IN 'NOTE'
                * DURATION COUNT IN 'LONG'
                * FIND 'UPTIME' AND 'DOWNTIME'
                * ACCORDING TO DUTY CYCLE CALLED
                * FOR BY 'VOICE'.
                *
D74E AD FF 02   LOOKUP LDA   NOTE ; GET NOTE NUMBER
D751 0A                ASL   ; DOUBLE IT
D752 A8                TAY
D753 B9 96 D7          LDA   NOTES,Y ; GET UPTIME
D756 85 00             STA   DOWNTIME ; SAVE IT
D758 AD FD 02          LDA   VOICE ; GET DUTY CYCLE
D75B 4A         SHIFT  LSR
D75C F0 04             BEQ   DONE ; SHIFT WIDTH COUNT
D75E 46 00             LSR   DOWNTIME ; ACCORDING TO VOICE
D760 D0 F9             BNE   SHIFT
D762 B9 96 D7   DONE   LDA   NOTES,Y ; GET ORIGINAL
D765 38                SEC
D766 E5 00             SBC   DOWNTIME ; COMPUTE DIFFERENCE
D768 85 01             STA   UPTIME ; SAVE IT
D76A C8                INY   ; NEXT ENTRY
D76B B9 96 D7          LDA   NOTES,Y ; GET DOWNTIME
D76E 65 00             ADC   DOWNTIME ; ADD DIFFERENCE
D770 85 00             STA   DOWNTIME
D772 A9 00             LDA   #0
D774 38                SEC
D775 ED FE 02          SBC   LONG ; GET COMPLIMENT OF DURATION
D778 85 03             STA   LENGTH+1 MOST SIGNIFICANT BYTE
D77A A9 00             LDA   #0
D77C 85 02             STA   LENGTH
D77E A5 01             LDA   UPTIME
D780 D0 98             BNE   PLAY ; IF NOT NOTE #0, PLAY IT
                *
                * 'REST' SUBROUTINE' PLAYS NOTE #0
                * SILENTLY, FOR SAME DURATION AS
                * A REGULAR NOTE.
                *
D782 EA         REST   NOP   ; DUMMY
D783 EA                NOP   ; CYCLE USERS
D784 4C 87 D7          JMP   REST2 ; TO ADJUST TIME
D787 E6 02      REST2  INC   LENGTH
D789 D0 05             BNE   REST3
D78B E6 03             INC   LENGTH+1
D78D D0 05             BNE   REST4
D78F 60                RTS   ; IF DURATION EXPIRED
D790 EA         REST3  NOP   ; USE UP 'INC' CYCLES
D791 4C 94 D7          JMP   REST4
D794 D0 EC      REST4  BNE   REST ; ALWAYS TAKEN
```

```
                    113 *
                    114 * NOTE TABLES
                    115 *
D796 00 00 F6       116 NOTES    HEX    00,00,F6,F6,E8,E8,DB,DB
D79E CF CF C3       117          HEX    CF,CF,C3,C3,B8,B8,AE,AE
D7A6 A4 A4 9B       118          HEX    A4,A4,9B,9B,92,92,8A,8A
D7AE 82 82 7B       119          HEX    82,82,7B,7B,74,74,6D,6E
D7B6 67 68 61       120          HEX    67,68,61,62,5C,5C,57,57
D7BE 52 52 4D       121          HEX    52,52,4D,4E,49,49,45,45
D7C6 41 41 3D       122          HEX    41,41,3D,3E,3A,3A,36,37
D7CE 33 34 30       123          HEX    33,34,30,31,2E,2E,2B,2C
D7D6 29 29 26       124          HEX    29,29,26,27,24,25,22,23
D7DE 20 21 1E       125          HEX    20,21,1E,1F,1D,1D,1B,1C
D7E6 1A 1A 18       126          HEX    1A,1A,18,19,17,17,15,16
D7EE 14 15 13       127          HEX    14,15,13,14,12,12,11,11
D7F6 10 10 0F       128          HEX    10,10,0F,10,0E,0F

--- END ASSEMBLY ---

TOTAL ERRORS: 00
```

APPENDIX II
SUMMARY OF PROGRAMMER'S AID COMMANDS

Chapter 1: RENUMBER

(a) To renumber an entire BASIC program:

```
CLR
START = 1ØØØ
STEP = 1Ø
CALL -1Ø531
```

(b) To renumber a program portion:

```
CLR
START = 2ØØ
STEP = 2Ø

FROM = 3ØØ                    (program portion
TO = 5ØØ                      to be renumbered)

CALL -1Ø521
```

Chapter 2: APPEND

(a) Load the second BASIC program, with high line numbers:

```
LOAD
```

(b) Load and append the first BASIC program, with low line numbers:

```
CALL -11Ø76
```

Chapter 3: TAPE VERIFY (BASIC)

(a) Save current BASIC program on tape:

```
SAVE
```

(b) Replay the tape, after:

```
CALL -1Ø955
```

Chapter 4: TAPE VERIFY (Machine Code and Data)

(a) From the Monitor, save the portion of memory on tape:

address1 . address2 W return

(b) Initialize Tape Verify feature:

D52EG return

(c) Replay the tape, after:

address1 . address2 ctrl Y return

Note: spaces shown within the above commands are for easier reading only; they should not be typed.

Chapter 5: RELOCATE (Machine Code and Data)

(a) From the Monitor, initialize Code-Relocation feature:

D4D5G return

(b) Blocks are memory locations from which program runs. Specify Destination and Source Block parameters:

Dest Blk Beg < Source Blk Beg . Source Blk End ctrl Y * return

(c) Segments are memory locations where parts of program reside. If first program Segment is code, Relocate:

Dest Seg Beg < Source Seg Beg . Source Seg End ctrl Y return

If first program Segment is data, Move:

Dest Seg Beg < Source Seg Beg . Source Seg End return

(d) In order of increasing address, Move subsequent contiguous data Segments:

. Source Segment End ctrl Y return

and Relocate subsequent contiguous code Segments:

. Source Segment End M return

Note: spaces shown within the above commands are for easier reading only; they should not be typed.

Chapter 6: RAM TEST

(a) From the Monitor, initialize RAM Test program:

D5BCG return

(b) To test a portion of memory:

address . pages ctrl Y return (test begins at address, continues for length pages.

Note: test length, pages*1ØØ, must <u>not</u> be greater than starting address. One page = 256 bytes ($1ØØ bytes, in Hex).

(c) To test more memory, do individual tests or concatenate:

addr1.pages1 ctrl Y addr2.pages2 ctrl Y addr3.pages3 ctrl Y return

Example, for a 48K system:

4ØØ.4 ctrl Y 8ØØ.8 ctrl Y 1ØØØ.1Ø ctrl Y 2ØØØ.2Ø ctrl Y
3ØØØ.2Ø ctrl Y 4ØØØ.4Ø ctrl Y 7ØØØ.2Ø ctrl Y 8ØØØ.4Ø
ctrl Y return

(d) To repeat test indefinitely:

N complete test 34:Ø type one space return

Note: except where specified in step (d), spaces shown within the above commands are for easier reading only; they should <u>not</u> be typed.

Chapter 7: MUSIC

(a) Assign appropriate variable names to CALL and POKE locations (optional):

MUSIC = -1Ø473
PITCH = 767
TIME = 766
TIMBRE = 765

(b) Set parameters for next note:

POKE PITCH, p (p = 1 to 5Ø; 32 = middle C)
POKE TIME, m (m = 1 to 255; 17Ø = 1 second)
POKE TIMBRE, t (t = 2, 8, 16, 32 or 64)

(c) Sound the note:

CALL MUSIC

Chapter 8: HIGH-RESOLUTION GRAPHICS

(a) Set order of parameters (first lines of program):

```
1  XØ = YØ = COLR
2  SHAPE = ROT = SCALE          (if shapes are used)
```

(b) Assign appropriate variable names to subroutine calling addresses (optional; omit any subroutines not used in program):

```
1Ø  INIT = -12288 : CLEAR = -12274 : BKGND = -11471
11  POSN = -11527 : PLOT = -115Ø6 : LINE = -115ØØ
12  DRAW = -11465 : DRAW1 = -11462
13  FIND = -1178Ø : SHLOAD = -11335
```

(c) Assign appropriate variable names to color values (optional; omit any colors not used in program):

```
2Ø  BLACK = Ø : LET GREEN = 42 : VIOLET = 85
21  WHITE = 127 : ORANGE = 17Ø : BLUE = 213
22  BLACK2 = 128 : WHITE2 = 255
```

(d) Initialize:

```
3Ø  CALL INIT
```

(e) Change screen conditions, if desired. Set appropriate parameter values, and CALL desired subroutines by name.

Example:

```
4Ø  COLR = VIOLET : CALL BKGND : REM  TURN BACKGROUND VIOLET
5Ø  FOR I = Ø TO 279 STEP 5
6Ø  XØ = 14Ø : YØ = 15Ø : COLR = WHITE : REM  SET PARAMETERS
7Ø  CALL POSN : REM  MARK THE "CENTER"
8Ø  XØ = I : YØ = Ø : REM  SET NEW PARAMETERS
9Ø  CALL LINE : REM  DRAW LINE TO EDGE
1ØØ  NEXT I : END
```

QUICK REFERENCE TO HIGH-RESOLUTION INFORMATION

Subroutine Name	CALLing Address	Parameters Needed
INIT	-12288	
CLEAR	-12274	
BKGND	-11471	COLR
POSN	-11527	XØ, YØ, COLR
PLOT	-115Ø6	XØ, YØ, COLR
LINE	-115ØØ	XØ, YØ, COLR
DRAW	-11465	XØ, YØ, COLR, SHAPE, ROT, SCALE
DRAW1	-11462	SHAPE, ROT, SCALE
FIND	-1178Ø	
SHLOAD	-11335	

Color Name	COLR Value	Color Name	COLR Value
BLACK	Ø	BLACK2	128
GREEN	42	ORANGE	17Ø
VIOLET	85	BLUE	213
WHITE	127	WHITE2	255

(Note: on systems below S/N 6ØØØ, colors in the second column appear identical to those in the first column)

CHANGING THE HIGH-RESOLUTION GRAPHICS DISPLAY

Full-Screen Graphics	POKE -163Ø2, Ø
Mixed Graphics-Plus-Text (Default)	POKE -163Ø1, Ø
Page 2 Display	POKE -16299, Ø
Page 1 Display (Normal)	POKE -163ØØ, Ø
Page 2 Plotting	POKE 8Ø6, 64
Page 1 Plotting (Default)	POKE 8Ø6, 32

(Note: CALL INIT sets mixed graphics-plus-text, and Page 1 plotting, but does not reset to Page 1 <u>display</u>.)

Collision Count for Shapes	PEEK (81Ø)

(Note: the change in PEEKed value indicates collision.)

apple computer inc.®
10260 Bandley Drive
Cupertino, California 95014
(408) 996-1010