PROGRAMMER'S AID #1

INSTALLATION AND OPERATING MANUAL

TM

Apple Utility Programs

Published by
APPLE COMPUTER INC.
10260 Bandley Drive
Cupertino. California 95014
(408) 996–1010

Reorder APPLE Product #A2L0011
(030–0026)

TABLE OF CONTENTS

INTRODUCTION

CHAPTER 1

RENUMBER

CHAPTER 2

APPEND

CHAPTER 3

TAPE VERIFY (BASIC)

8 VerifyIng a Basic program saved on tape
8 Comments

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 #l 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 am 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 6502 machine—language programs.

Chapter 6. Test the memory of the APPLE.

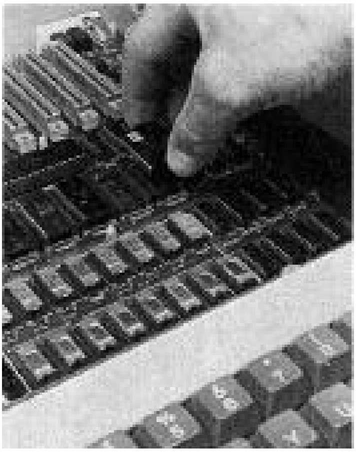
Chapter 7. Generate musical notes of variable duration over four chromatic octaves, in five (slightly) different timbres, from Integer BASIC.

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—D0”.

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 D0, 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 he 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 1000. Type

START = 1000

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 remumber in increments of 10, type

STEP = 10

Finally, type the this commands

CALL —10531

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—>1000
213—>1010
527—>1020
698—>1030
13000—>1040
13233—>1050

RENUMBERING PORTIONS OF A PROGRAM

You do not have to renumber your entire program. You can renumber just the lines numbered from, say, 300 to 500 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, <u>after</u> renumbering, will begin with line number 200. say, and that each line's number thereafter will be 20 greater than the previous line's:

START = 200
STEP = 20

The next two variables specify the program portion's range of line numbers <u>before</u> renumbering.

FROM = 300
TO = 500

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

CALL —10521

If the program was previously numbered

```
 100
 120
 300
 310
 402
 500
2000
2022
```

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

```
300—>200
310—>220
402—>240
500—>260
```

and the new program line numbers will be

```
 100
 120
 200
 220
 240
 260
2000
2022
```

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 nay result in loss of the program.

3. If am 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 10+20 will not be renumbered correctly.

4.Nonsense values for STEP, such as 0 or a negative number, can render your program unusable. A negative START value cam 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 cam 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 teen 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 he 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 —11076

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 —11059

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 programs 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 tine 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 -10955

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. Amy 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 routine 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 tine the tape is verified, will cause the verification to fail.

CHAPTER 5
RELOCATE

PART A: THEORY OF OPERATION

LOCATING MACHINE-LANGUAGE CODE

Quite frequently. programmers encounter situations that call -for relocating machine-language (not BASIC) programs on the 6502-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 snail 6502 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 dissembler, 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. No 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 nay have to be performed on the smaller machine.

5. Because of memory- mapping differences, a program that ran on an APPLE I (or other 6502-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.

10. On a whim, the user wishes to move a program.

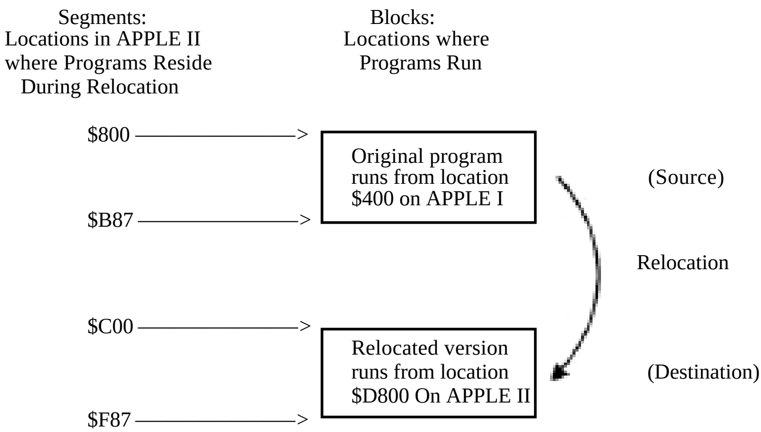
PROGRAM MODEL

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 $400 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 sane 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 $400 may currently reside beginning at -location $800. 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 $D800 (a ROM address) yet reside beginning at location $C00 prior to being saved on tape or used to burn EPROMs (obviously, the relocated program cannot immediately reside at locations reserved for RON). In some cases, the Source and Destination Segments may overlap.

BLOCKS AND SEGMENTS EXAMPLE



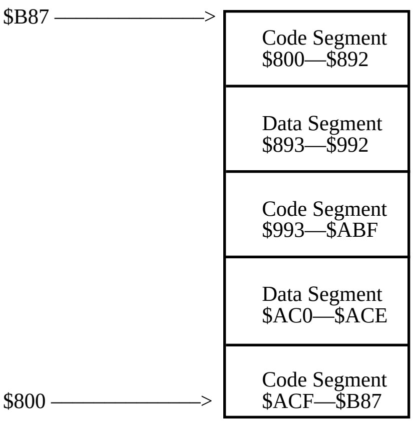
SOURCE BLOCK $400-$787: DESTINATION BLOCK: $D800—$DB87

SOURCE SEGMENTS.: $800—$B87 DESTINATION SEGMENTS: $C00—$F87

DATA SEGMENTS

The problem with relocating a large program all at once is that blocks of data (tables, text, etc.) nay be interspersed throughout the code. During relocation, this date 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 6502 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-ReIocatIon 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 N 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)

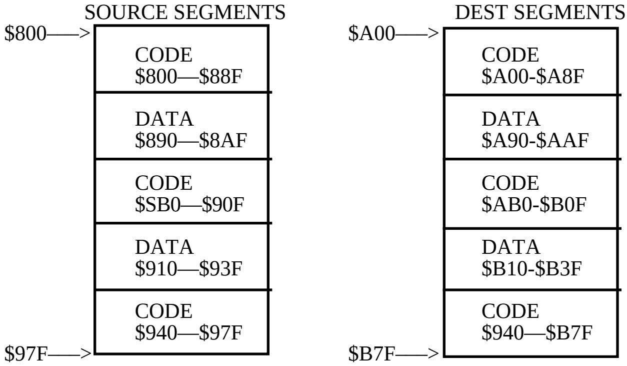
Note: the shortened form of the command cam only be used if each "subsequent" segment is contiguous to the segment previously relocated or Moved. If a "subsequent" segment is in a part of memory that does not begin exactly where the previous segment ended, it must be Moved or relocated using the full "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 $800—$97F. The relocated version will reside and run in locations $A00—$B7F.



SOURCE BLOCK: $800—$97F   DEST BLOCK: $A00-$B7F
SOURCE SEGMENTS: $800—$97F   DEST SEGMENTS: $A00-$B7F

(a) Initialize Code—Relocation feature:

reset D4D5G return

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

A00 < 800 - 97F ctrl Y * return

(C.) Relocate first segment (code):

A00 < 800 .88F ctrl Y return

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

• 8AF	M return	(data)
• 90F	ctrl Y return	(code)
• 93F	M return	(data)
• 97F	ctrl Y return	(code)

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

A90 < 890 • 8AF M return	(data)
AB0 < 8B0 • 90F ctrl Y return	(code)
B10 < 910 • 93F M return	(data)
B40 <940 • 97F ctrl Y return	(code)

EXAMPLE 2. Index into Block

Suppose that the program of Example I uses an indexed reference into the Data Segment at $890 as follows:

LDA 7B0,X

where the X-REG is presumed to contain a number in the range $E0 to $FF. Because address $730 is outside the Source Block, it will not he relocated. This nay be handled in one of two ways.

(a) You. nay fix the exception by hand; or

(b) You nay 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:

900 < 700 . 97F ctrl Y * return

Note: with this Block specification, <u>all</u> program references to the "prior page" (in this case the $700 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  LOC0
LDA  #$08
STA  LOC1
JMP  (LOC0)
```

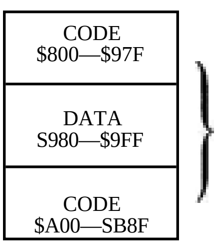
In this example, the LDA #$08 will not be changed during relocation and the user will have to hand-modify it to $0A.

EXAMPLE 4. Unusable Block Ranges

Suppose a program was written to run from locations $400-$78F on an APPLE 1. A version which will run in ROM locations SD800-SDB8F must be generated. The Source (and Destination) Segments will reside in locations $800—$B8F on the APPLE II during relocation.

Addresses during relocation — Source And Destination Segments — Source And Destination Blocks

$800———>



$B8F ———>

Rums from locations $400-$78F on an APPLE 1, but must be relocated to run from locations $D800-$DB8F on the APPLE II.

SOURCE BLOCK: $400-$78F: DEST BLOCK: $D800—$DB8F:
SOURCE SEGMENTS: $800-$B8F DEST SEGMENTS: $800—SB8F

(a) Initialize the Code-Relocation feature:

reset D4D5G return

(b) Load original program into locations $800—$B8F (despite the fact that it doesn't run there):

800 . B8F R return

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

0800 < 400 . 78F ctrl Y return

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

800 < 800 . 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 ($800) and need not be specified. The initial segment relocation command may be abbreviated as follows:

800 < . 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 $20 to $3F. Because these locations are reserved for the APPLE II system monitor, the allocation must be changed to locations $80—$9F. The Source and Destination Blocks are thus not the program but rather the variable area.

SOURCE BLOCK:	$20-$3F	DEST BLOCK:	$80-$9F
SOURCE SEGMENTS:	$S00-$97F	DEST SEGMENTS:	$800-$97F

(a) Initialize the Code-Relocation feature:

reset D4D5G return

(b) Specify Destination and Source Blocks:

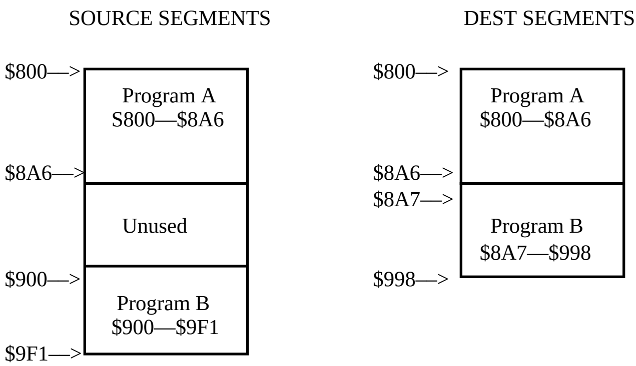
80 < 20 . 3F ctrl Y * return

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

800 < . 88F ctrl Y return	(first segment, code)
. 8AF M return	(data)
. 90F 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 $800—$8A6. Program B resides and runs in locations $900—$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 BLOCK:	$900-$9F 1	DEST BLOCKS:	$8A7-$998
SOURCE SEGMENTS:	$800-$8A6 (A)	DEST SEGMENTS:	$800-$8A6 (A)
	$900-$9F1 (B)		$8A7-$998 (B)

(a) Initialize the Code-Relocation feature:

04B5G return

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

8A7 < 900 . 9F1 ctrl Y * return

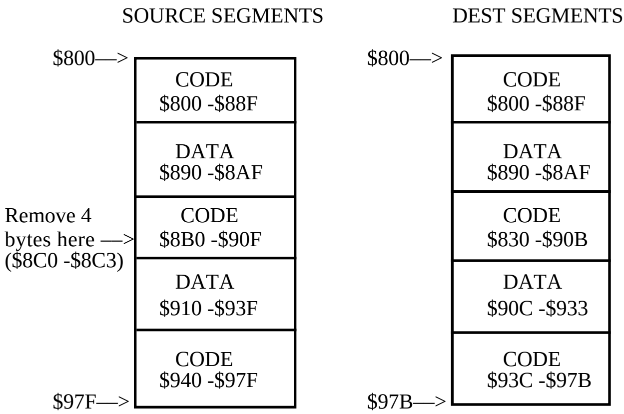
(c) Relocate each of the two programs individually. Program A must be relocated even though it does not move.

800 < . 8A6 ctrl Y return (program A, “in place”)
8A7 < 900 . 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:	$8C0 -$97B
SOURCE SEGMENTS:	$800 -$88F (code)	DEST SEGMENTS:	$800 -$88F (code)
	$890 -$8AF (data)		$890 -$8AF (data)
	$8B0 -$8BF (code)		$8B0 -$8BF (code)
	$8C4 -$90F (code)		$8C0 -$90B (code)
	$910 -$93F (data)		$90C -$93B (data)
	$940 -$97F(code)		$93C -$97B(code)

(a) Initialize Code-Relocation feature:

reset D4D5G return

(b) Specify Destination and Source Blocks:

8C0 < 8C4 . 97F ctrl Y* return

(e) Relocate Code Segments and Move Data Segments, in ascending address Sequence

800 < . 88F ctrl Y return	(first segment, code, “in place”)
. 8AF M return	(data)
. 8BF ctrl Y return	(code)
8C0 < 8C4 . 90F 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: $F700 -$FFFF   DEST BLOCK: $700 -$FFF
(see EXAMPLE 2)

SOURCE SEGMENTS:	$F800 -$F961 (code)	DEST SEGMENTS:	$800—$961 (code)
	$F962 -$FA42 (data)		$962 -$A42 (data)
	$FA43 -$FB18 (code)		$A43 -$B 18 (code)
	$FB19 -$FBlD (data)		$319 -$B1D (data)
	$FB1E -$FFCB (code)		$B1E -$FCB (code)
	$FFCC -$FFFF (data)		$FCC -$FFF (data)

IMMEDIATE ADDRESS REFERENCES (see EXAMPLE 3) $F FBF
$FEA8
(more if not relocating to page boundary)

(a) Initialize the Code—Relocation feature:

reset D4D5G return

(b) Specify Destination and Source Block parameters:
700 < F700 . FFFF ctrl * return

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

800 < F800 . 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 < Addrl . Addr2 ctrl Y return (Addr1, Addr2, and Addr4 are addresses)

vectors to location $3F8 with the value Addrl 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 $200.

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	ERMEND

3. If ctrl Y is not followed by * , then a segment relocation is initiated at RELOC2 ($3BB). Throughout, Addrl ($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 6502 instruction, given the opcode in the A-REG, and stores that opcode's instruction length in the variable LENGTH (location $2r)

Instruction Type in A-REG	LENGTH (in $2F)
Invalid	0
1 byte	0
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 6502 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) them go to step 7.

5. If two-byte: instruction then clear 3rd byte so address field is 0-255 (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 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 (4096) 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 tine, 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 labeled "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 dust 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 ($100 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*100 must not be greater than starting address a.

For example,

2000.10 ctrl Y return

tests hexadecimal 1000 bytes of memory (4096, or "4K" bytes, in decimal), starting at hexadecimal address 2000 (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	0000—0FFF	4K
4K	D	1000—IFFF	8K
4K	E	2000—2FFF	12K
16K	C	0000—3FFF	16K
4K	D	4000—4FFF	20K
4K	E	5000—5FFF	24K
16K	C	0000—3FFF	16K
16K	D	4000—7FFF	32K
16K	E	8000—BFFF	48K

A 4K RAM Row contains 10 Hex pages (hex 1000 bytes, or decimal 4096 bytes).
A 16K RAM Row contains 40 Hex pages (hex 4000 bytes, or decimal 16384 bytes).

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

400.4 ctrl Y return <——This tests the screen area of memory
800.8 ctrl Y return These first four tests examine
1000.10 ctrl Y return <—— the first 16K row of memory (Row C)
2000.20 ctrl Y return
4000.40 ctrl Y return <—— This tests the second 16K row of memory (Row D)
8000.40 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:

3000.20 ctrl Y return

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

7000.20 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:

400.4 ctrl Y 800.8 ctrl Y 1000.10 ctrl Y 2000.20 ctrl Y 3000.20 ctrl Y 4000.40 ctrl Y 7000.20 ctrl Y 8000.40 ctrl Y return

Remember, ctrl Y will not print on the screen, but it must be typed. With the single exception noted in the section TESTING FOR INTERMITTENT FAILURE, spaces are shown for easier reading but should not 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 50 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 same 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;
z z 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 10.

Example:

201F 00 10 ERR D-I

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 10. In this type of error, the indicated row (but not the column) may he 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:

5051 00 08 5451 00 ERR E-6

After a dynamic error, the indicated row (but not the column) may he 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 34:0
c. type at least one space before pressing the RETURN key.

Here is the format:

.N (memory test to be repeated) 34:0 (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 400.4 ctrl Y 800.8 ctrl Y 1000.10 ctrl Y 2000.20 ctrl Y 3000.20 ctrl Y 4000.40 ctrl Y 7000.20 ctrl Y 8000.40 ctrl Y 34:0 return

Run this test for at least one ho,~r (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 tines; 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 400 (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 100 Hex. 2000.30 ctrl Y will not work, but 5000.30 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 Memory Expansion-Module (P/N A2M0014). To add 16K of memory, order the 16K Memory Expansion Module (P/N A2M0016).

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 sane 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 =-10473
PITCH = 767
TIME = 766
TIMBRE = 765
```

There are 50 notes available, numbered from 1 to 50. 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 0 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 = 170 gives notes that are approximately one second long. To get notes at. a metronome marking of MM, use a duration of 10200/MM. For example, to get 204 notes per minute (approximately) use the command

```
POKE TIME, 10200/204
```

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 TINE, 40: POKE TIMBRE, 32
30 FOR I = 1 TO 49
40 POKE PITCH, I
50 CALL MUSIC
60 NEXT I: END
```

Where K 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 If 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 6000), displayed from two different "pages" of memory. The standard low-resolution graphics allowed you to plot 40 squares across the screen by 47 squares from top to bottom of the screen. This high-resolution graphics display node lets you plot in much smaller dots, 280 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 10.5K 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:

```
0    X0 = Y0 = 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 0 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 0. 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 XO, but X or XCOORD could] not

The parameters SHAPE. ROT, and SCALE are used only by the subroutines that draw shapes (DRAW and DRAWl, 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 tine 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   :  DRAWl = - 11462
8    FIND  = - 11780  :  SULOAD =- 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.

```
0 X0 = YO = COLR = SHAPE = ROT = SCALE
5 INIT =- 12288k : PLOT = -11506 : DRAW = -11465
10  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:

```
0  X0  =  Y0  =  COLR  =  SHAPE  =  ROT  =  SCALE
2  l = J = K
5  INIT =- 12288 : CLEAR  = - 12274
6  BKGND  =- 11471 : POSN  = - 11527
10  BLACK  = 0 : 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, BKCND, POS, 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 280 x-positions wide (X0=0 through X0=279) by 160 y-positions high (Y0=0 through Y0=l59), with an area for four lines of text at the bottom of the screen. Y0 values from 0 through 191 may be used, but values greater than 159 will not be displayed on the screen. The graphics origin (X0=0, Y0=0) is at the top left corner of the screen.

CHANGING THE GRAPHICS SCREEN

If you wish to devote the entire display to graphics (280 x-positions wide by 192 y-positions high), use

POKE -16302, 0

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

POKE -16301, 0

or another

CALL INIT

When the High-Resolution subroutines are first initialized, all graphics are done in Page 1 of memory ($2000-3FFF), and only that page of memory is displayed. If you wish to use memory Page 2 (S4000-5FFF), two POKEs allow you to do so:

POKE 806, 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, 0

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

POKE -16300, 0

while

POKE 806, 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 corn to the background color you desire, and then

CALL BKGND

The following program turns the entire display violet:

```
0    X0  =  Y0  =  COLR : REM SET PARAMETERS
5    INIT =- 12288 : BKGND = -11471 : REM DEFINE SUBROUTINES
10   VIOLET = 85 : REM DEFINE COLOR
20   CALL INIT : REM INITIALIZE HIGH-RESOLUTION SUBROUTINES
30   COLR = VIOLET : REM ASSIGN COLOR VALUE
40   CALL BRGND : REM MAKE ALL OF DISPLAY VIOLET
50   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 X~, Y0 and COLR. KO must in the range from 0 through 279, YO must be in the range from 0 through 191, and COLR must be in the range from 0 through 255, or a

*** RANGE ERR

message will be displayed and the program will halt.

The program below plots a white dot at K-coordinate 35, Y-coordinate 55, and a violet dot at K-coordinate 85, Y-coordinate 90:

```
0   X0 = COLR : REM SET PARAMETERS
5   INIT = —12288 : PLOT =- 11506 : REM DEFINE SUBROUTINES
10   WHITE = 127 : VIOLET = 85 : REM DEFINE COLORS
20   CALL INIT :  REM INITIALIZE SUBROUTINES
30   COLR = WHITE :REM ASSIGN PARAMETER VALUES
40   X0 = 35 : Y0 = 55
50   CALL PLOT : REM PLOT WITH ASSIGNED PARAMETER VALUES
60   COLR = VIOLET : REM ASSIGN NEW PARAMETER VALUES
70   X0  =  85 : Y0 = 90
80   CALL PLOT   REM PLOT WITH NEW PARAMETER VALUES
90   END
```

The subroutine POSN is exactly like PLOT, except that nothing is placed on the screen. COLE must be specified, however, and a subsequent DRAWl (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 X0, Y0, 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, X0 and Y0 (the coordinates of the termination point for the line), and COLE must be assigned legitimate values, or an error nay occur, The following program draws a grid of green lines vertically and violet lines horizontally, on a white background:

```
0    X0 = Y0 = COLR : REM SET PARAMETERS. THEN DEFINE SUBROUTINES
5    INIT =- 12288 : BKGND  = - 11471 : PLOT =- 11506 : LINE = - 11500
10   LET GREEN = 42 : VIOLET = 85 : WHITE = 127 : REM DEFINE COLORS
20    CALL INIT : REM INITIALIZE HIGH-RESOLUTION SUBROUTINES
30    POKE - 16302, 0 : REM SET FULL-SCREEN GRAPHICS
40    COLR = WHITE : CALL BKGND : REM MAKE THE DISPLAY ALL WHITE
50    COLR = GREEN : REM ASSIGN PARAMETER VALUES
60    FOR X0 = 0 TO 270 STEP 10
70    Y0 = 0 : CALL PLOT : REM PLOT A STARTING-POINT AT TOP OF SCREEN
80    Y0 = 190 : CALL LINE : REM DRAW A VERTICAL LINE TO BOTTOM OF SCREEN
90    NEXT X0 : REM MOVE RIGHT AND DO IT AGAIN
100  COLR = VIOLET : REM ASSIGN NEW PARAMETER VALUES
110  FOR Y0 = 0 10 190 STEP 10
120  X0 = 0 : CALL PLOT : REM PLOT A STARTING-POINT AT LEFT EDGE OF SCREEN
130  X0 = 270 : CALL LINE : REM PLOT A HORIZONTAL LINE TO RIGHT EDGE
140  NEXT Y0 : REM MOVE DOWN AND DO IT AGAIN
150  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 nay be created, used, and saved in a “Shape Table”, through the use of the High-Resolution subroutines DRAW, DRAWl 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 DRAWl (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	0
Specifies:	D	D	P	D	D	P	D	D

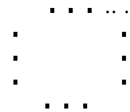
Each bit pair DD specifies a direction to move, and each bit P specifies whether or not to plot a point before moving, as follows:

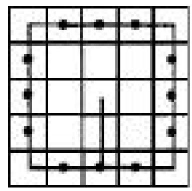
If DD	= 00	move up		
	= 01	move right	If P = 0	don’t plot
	= 10	move down	=1	do plot
	= 11	move left		

Notice that the last section, C (the two most significant bits), does not have a P field (by default, P=0), so section C can only specify a move with<u>out</u> plotting.

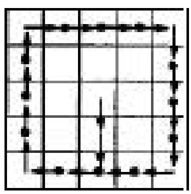
Each byte can represent up to three plotting vectors, one in section A, one in section B. and a third (a move only) in section C.

DRAW and DRAWl 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 00 (a move up, without plotting) because that section, containing only zeros, will be ignored. Similarly, if section C is 00 (ignored), then section B cannot be a move of 000 as that will also be ignored. And a move of 000 in section A will <u>end</u> your shape definition unless there is a 1-bit somewhere in section II 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 90 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:

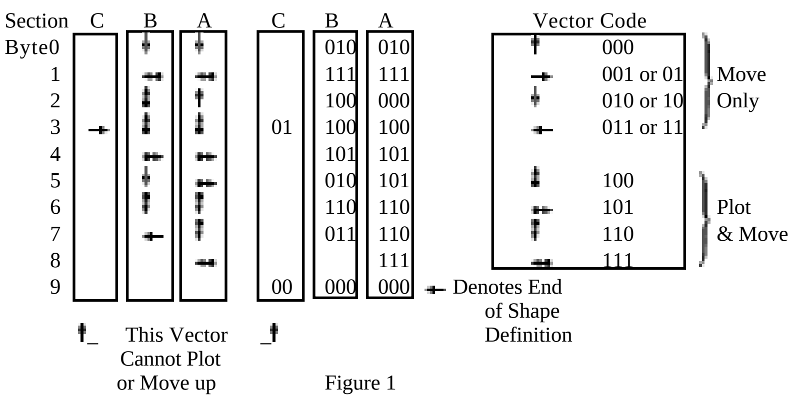


Figure 1

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 00 (or 000) 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	B	A		Bytes Recoded in Hex	
Byte 0	0 0	0 1	0 0 1 0	=	1 2	
1	0 0	1 1	1 1 1 1	=	3 F	
2	0 0	1 0	0 0 0 0	=	2 0	
3	0 1	1 0	0 1 0 0	=	6 4	
4	0 0	1 0	1 1 0 1	=	2 D	
5	0 0	0 1	0 1 0 1	=	1 5	
6	0 0	1 1	0 1 1 0	=	3 6	
7	0 0	0 1	1 1 1 0	=	l E	
8	0 0	0 0	0 1 1 1	=	0 7	
9	0 0	0 0	0 0 0 0	=	0 0	Denotes End of Shape Definition
Hex:	Digit 1		Digit 2			

Figure 2

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

Figure 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 0 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 (00 04, in hexadecimal). Therefore, index byte S+2 must contain the value 04 and index byte S+3 must contain the value 00. The completed Shape Table for this example is shown in Figure 5 on the next page.

Start=S →	Byte S+0	n (0 to FF)	← Total Number of Shape Definitions
Index	+1	Unused	
	+2	Lower 2 Digits	← Dl: Index to First Byte of Shape Definition #1, Relative to S
	+3	Upper 2 Digits	
	+4	Lower 2 Digits	← D2: Index to First Byte of Shape Definition #2, Relative to S
	+5	Upper 2 Digits	
	. . .	. . .	
	+2n	Lower 2 Digits	← Dn: Index to First Byte of Shape Definition #n, Relative to S
	+2n+1	Upper 2 Digits	
Shape Definitions	S+D1	First Byte	← Shape Definition #1
	.	. .	
	.	Last Byte	
	S+D2	First Byte	← Shape Definition #2
	.	. .	
	.	Last Byte	
	. . .	. . .	
	S+Dn	First Byte	← Shape Definition #n
	.	. .	
	.	Last Byte+00	

Figure 4

Start → (Store this address in $328 and $329)	Byte 0	01	← Number of Shapes
	1	00	
	2	04	← Index to Shape Definition #1, Relative to Start
	3	00	
	4	12	← First Byte
	5	3F	
	6	20	
	7	64	
	8	2D	← Shape Definition #1
	9	15	
	A	36	
	B	1E	
	C	07	
	D	00	← Last Byte

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 0800.

Note: this address must he 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 $2000 to $4000) or Page 2 (hexadecimal locations $4000 to $6000) for high-resolution graphics. Furthermore, it must not be in an area of memory used by your BASIC program. Hexadecimal 0800 (2048, 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:

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:

0800 return

(type "return'~ by pressing the RETURN key). What does APPLE say the contents of location 0800 are? Now try this:

```
 0800:01 return
 0800 return
0800—  01
```

The APPLE now says that the value 01 (hexadecimal) is stored in the location whose address is 0800. To store more two-digit hexadecimal numbers in successive bytes in memory, just open the first address:

and then type the numbers, separated by spaces:

0800:01 00 04 00 12 3F 20 64 2D 15 36 IE 07 00 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:

```
0800 return
0800-  01
return
00 04 00 12 3F 20 64
return
0808—          2D 15 36 1E 07 00 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 cam 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 08 00, this would do the trick:

```
328:00 08
```

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 3 return) to decimal 2048 (0800. im hexadecimal). You must then change LOMEM to 2048 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:      2048 + 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 (0800. in our example)
2) Last Address of the table (080D, in our example)
3) Difference between 2) and 1) (000D, in our example)

Item 3, the difference between the last address and the first address of the table. must be stored in hexadecimal locations 0 (lower two digits) and 1 (upper two digits):

0:0D 00 return

Now you can "Write" (store on cassette) first the table length that is stored in locations 0 and 1, and then the Shape Table itself that is stored in locations Starting Address through Last Address:

0.1W 0800.080DW

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 format

BSAVE filename. A$ startingaddress, L$ tablelength

For our example, you might type

BSAVE MYSHAPE1, AS 0800. LS 000D

Note: the Disk Operating System (DOS) occupies the top 10.5K of memory (10752 bytes decimal, or $2A00 hex); make sure your Shape Table is not in that portion of memory when you "boot" the disk system.

LOADING A SHAPE TAIL!

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 0800. 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, MYSHAPEL would BLOAD beginning at address 0800. You must store the Shape Table’s starting address in hexadecimal locations 328 mmd 329, yourself, from the Monitor:

328:00 08 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: 2048 + 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 = COLE = SHAPE = ROT = SCALE REM SET PARAMETERS
20   INIT = -12288 : DRAW —11465 REM DEFINE SUBROUTINES
30   WRITE = 127 : BLACK = 0 : REM DEFINE COLORS
40   CALL INIT : REM INITIALIZE HIGH-RESOLIJTION 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 draw, 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 I: DRAWING SHAPES FROM A PREPARED SHAPE TABLE

before either of the two shape-drawing subroutines DRAW or DRAWl 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 (808 and 809, im 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 X0 and Y0. 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 0 through 255, but SCALE = 0 is interpreted as SCALE = 256, the largest 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 = 0

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 90 degrees clockwise. The value assigned to ROT must be within the range 0 to 255 (although ROT=64, specifying a rotation of 360 degrees clockwise, is the equivalent of ROT=0). For SCALE=1, only four of the 63 different rotations are recognized (0.16,32,48); 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 = 0 (no rotation from shape definition)

ROT = 48 (270 degrees clockwise rotation)

ROT = 16 (90 degrees clockwise rotation)

ROT = 32 (180 degrees clockwise rotation)

DRAWING SHAPES

The following example program DRAWs shape definition number three. im white. at a 135 degree clockwise rotation. Its starting point, or origin, is at (140,80).

```
0  X0 = Y0 = COLR = SHAPE = ROT - SCALE : REM SET PARAMETERS
5  INIT=-12288 : DRAW = -11465 : REM DEFINE SUBROUTINES
10  WHITE = 127 : REM DEFINE COLOR
20  CALL INIT : REM INITIALIZE HIGH-RESOLUTION SUBROUTINES
30  X0 = 140 : Y0 = 80 : COLR = WHITE   REM ASSIGN PARAMETER VALUES
40  SHAPE = 3 : ROT =  24 : SCALE = 2
50  CALL DRAW : REM DRAM SHAPE 3, DOUBLE SIZE, TURNED 135 DEGREES
60  END
```

LINKING SHAPES

DRAWl 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. X0, TO. and COLE, need not be specified, as they will have no effect on DRAWl. 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 #0. then linking a new shape to the old one, each tine the game control gives a new orientation. To clear the screen of “squiggles,” press the game-control button.

```
10    X0 = Y0 = COLR = SHAPE = ROT = SCALE REM SET PARAMETERS
20    INIT = -12288 DRAW = -11465 DRAWl = -11462
22    CLEAR = -12274 UNITE = 127 REM NAME SUBROUTINES AND COLOR
30    FULLSCREEN = -16302 BUTN =-16287 REM NAME LOCATIONS
40    CALL INIT REM INITIALIZE HIGH-RESOLUTION SUBROUTINES
50    POKE FULLSCREEN, 0 REM SET FULL-SCREEN GRAPHICS
60    COLR = WHITE : SHAPE = 1 : SCALE = 5
70    X0 = 140 Y0 = 80 : REM ASSIGN PARAMETER VALUES
80    CALL CLEAR : ROT = PDL(0) : CALL DRAW : REM DRAW FIRST SHAPE
90    IF PEEK(BUTN) > 127 THEN GOTO 80 : REM PRESS BUTTON TO CLEAR SCREEN
100   R = PDL(0) : IF (R < ROT+2) AND (R >ROT+2) THEN GOTO 90 :
      REM WAIT FOR CHANGE IN GAME CONTROL
110 ROT = R : CALL DRAWl : REM ADD TO 'SQUIGGLE”
120 GOTO 90 : REM LOOK FOR ANOTHER CHANCE
```

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 DRAWS, 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 (10, 25).

```
0  X0 = Y0 = COLR = SHAPE = ROT = SCALE : REM SET PARAMETERS
5  INIT = -12288 : LINE = -11500 : DRAW = -11402 : FIND = -11780
10   VIOLET = 85 : WHITE = 127 : REM DEFINE SUBROUTINES AND COLORS
20   X0 = 140 : Y0 = 80 : COLR = WHITE : REM ASSIGN PARAMETER VALUES
30   SHAPE = 3 : ROT = 0 : SCALE = 2
40   CALL DRAW : REM DRAW SHAPE WITH ABOVE PARAMETERS
50   CALL : FIND REM FIND COORDINATES OF LAST SHAPE POINT
60   X0 = 10 : Y0 = 25 :COLR = VIOLET REM NEW PARAMETER VALUES, FOR LINE
70   CALL LINE : REM DRAW LINE WITH ABOVE PARAMETERS
80   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 DRAWL subroutines return a “collision count” in the hexadecimal memory location $32A (810. 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.

```
110 CALL DRAW : REM DRAW THE SHAPE
120 COUNT = PEEK(810) : REM FIND THE COLLISION COUNT
```

PART F: TECHNICAL INFORMATION

LOCATIONS OF THE HIGH-RESOLUTION PARAMETERS

When the high-resolution parameters are entered (line 0, 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 111gb—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

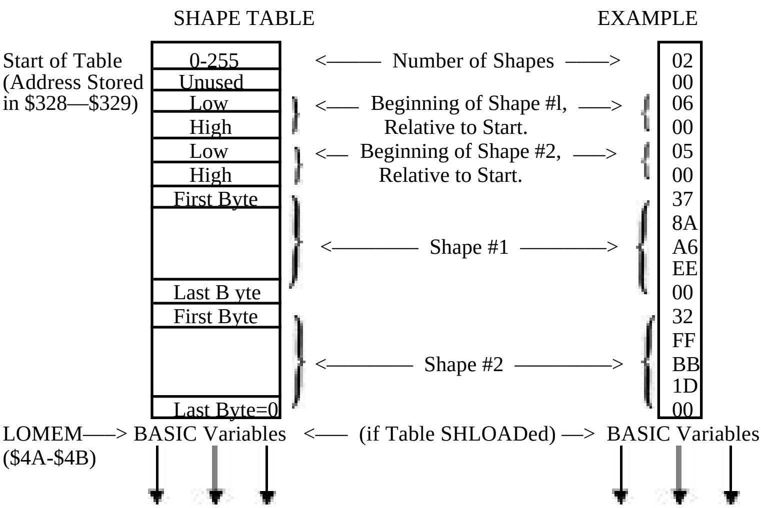
Parameter	Locations beyond LOMEM
X0	$05, $06
Y0	$0C, S0D
COLR	$15, $16
SHAPE	$IF, $20
ROT	$27, $28
SCALE	$31, $32

VARIABLES USED WITHIN THE HIGH-RESOLUTION SUBROUTINES

Variable Name	Hexadecimal Location	Description
SHAPEL, SHAPER	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	30	On-the-fly BIT MASK
QDRNT	53	2 LSB's are rotation quadrant for DRAW.
X0L. X0R	320, 321	Most recent X-coordinate. Used for initial endpoint of LINE. Updated by PLOT. POSN, LINE and FIND, not DRAW.
Y0	322	Most recent y-coordinate (see X0L,
BXSAV	323	Saves 6502 K-register during high- resolution CALLs from BASIC.
BCOLOR	324	Color specification for PLOT. POSN.
HNDX	325	On-the-fly byte index from BASES ADDRESS
HPAG	326	Memory page for plotting graphics. Normally ~20 for plotting in Page 1 of high—resolution display memory ($2000—$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, DRAWl.

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).

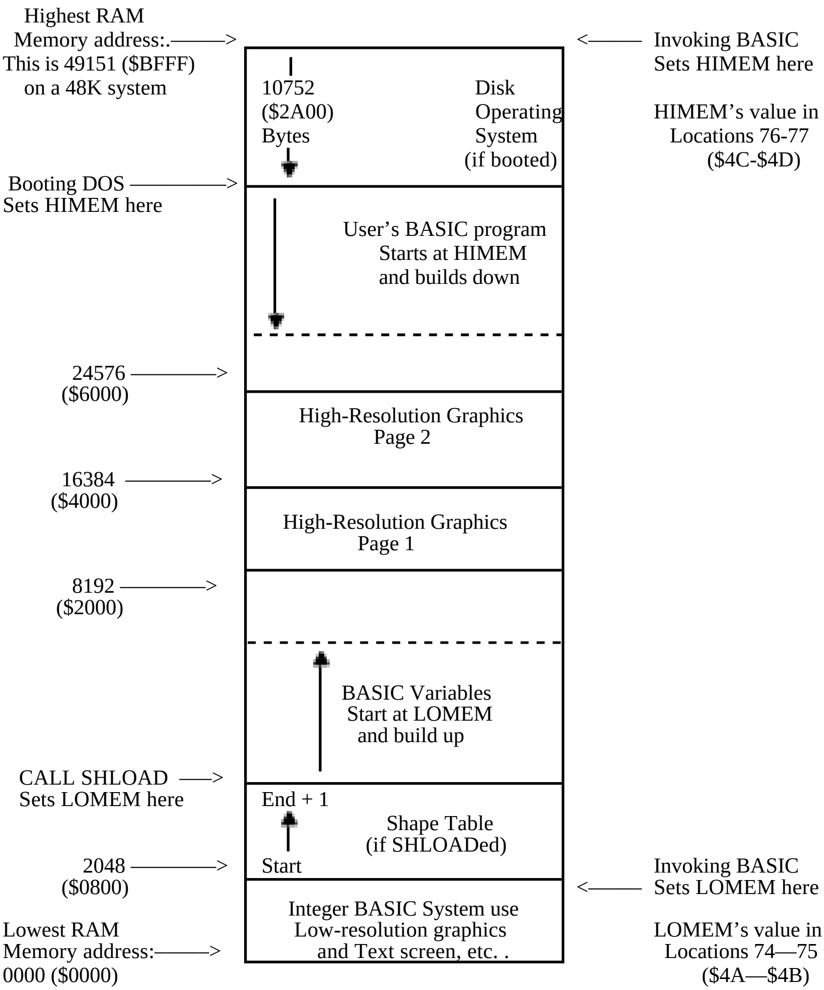


The address of the Shape Table’s Start should be stored in locations $328 and $329. If the SHLOAD subroutineis used to load the table. start will be set to LOMEM(normally this is at $0800) and then LOMEM will be moved to one byte after the end of the Shape Table, automatically.

If you wish to load a Shape Table named MYSIIAPES2 from disk, beginning at decimal location 2048 (0800 hex) and ending at decimal location 2048 plus decimal 15 bytes (as in the example above), you may wish to begin your BASIC program as follows:

```
0  D$ = ““ : REM QUOTES CONTAIN CTRL D (D$ WILL BE ERASED BY SHAPE TABLE)
1  PRINT D$; “BLOAD MYSHAPES2, A 2048” : REM LOADS SHAPE TABLE
2  POKE 808, 2048 MOD 256 POKE 809, 2048 / 256  :REM SETS TABLE START
3  POKE 74, (2048 + 15 + 1) MOD 256 POKE 75. (2048 + 15 + 1) / 256
4  POKE 204, PEEK(74) POKE 205, PEEK(75) : REM SETS LOMEM To TABLE END+l
5  X0 = Y0 = COLR = SHAPE = ROT = SCALE : REM SETS PAEM4ETERS
```

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



Unfortunately, there is no convention for napping memory. This map shows the highest (largest) address at the top, lowest (smallest) address at the bottom. The naps of Shape Tables that appear on other pages show the Starting address (lowest and smallest) at the top, the Ending address (highest end 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 ~f 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 cam usually use the DOS and both high-resolution memory pages without problems.

4. If you loaded your Shape ~able 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 any variables. SHLOAD does this automatically, but you can use this immediate-execution command:

LOMEM: 2048 + tablelength + 1

where tablelength must be a number, not 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        *
            *                            *
            ************t*****************.

            * 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   $SC     MONITOR Al.
            A1H        EQU   $3D
            A2L        EQU   $3E     MONITOR A2.
            A2H        EQU   $3F
            LOMEML     EQU   $4A.    BASIC ‘START CE VARS'.
            LOMEMH     EQU   $4B
            DXL,       EQO   $50     DELTA-X FOR HI IN, SHAPE.
            DXH.       EQU   $51
            SHAPEX     LQU   $51     SHAPE TEMP.
            DY         EQU   $52     DELTA-Y FOR HLIN. SHAPE.
            QDRNT      EQU   $53     ROT QUADRANT (SHAPE),
            EL         EQU   $54     ERROR FOR HLIN.
            EN         EQU   $55
            PPL        EQU   $CA     BASIC START OF PROG PTR.
            PPH        EQU   $CB
            PVL        EQU   $CC     BASIC END OF VARS PTR.
            PYH        EQU   $CD
            ACL        EQU   $CE     BASIC ACC.
            ACH        EQU   $CF
            X0L        EQU   $320    PRIOR X-COORD SAVE
            X0H        EQU   $321    AFTER HLIN OR HPLOT.
            Y0         EQU   $322    HLIN, HPLOT Y-COORD SAVE.
            BXSAV      EQU   $323    X-REG SAVE FOR SASIC.
            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.
            SWAP XL    EQU   $328    START OF
            SHAPXH     EQU   $329    -  SHAPE TABLE.
            COLLSN     EQU   $32A    COLLISION COUNT
            HIRES      EQU   $C057   SWITCH TO HI-RES VIDEO
            MIXSET     EQU   SC053   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   $E11E   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
            *.
            * RUM VERSION $D000 TO $D3FF
            *
                       ORG   $D000
                       ODJ   $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  IC          BKGNDOSTA       HCOLOR1 SET FOR BLACK BKGND.
D012  AD  26  03      BKGND     LDA   HPAG
D015  85  18                    STA   SHAPEH INIT HI-RES SCREEN MEM
D017  A0  00                    LDY   #$0 FOR CURRENT PACE, NORMALLY
D019  84  IA                    STY   SHAPEI. $2000-3FFF OR $4000-5FFF
D01B  A5  1C          BKGND1    LDA   HCOLOR1
D01D  91  1A                    STA   (SHAPEL) Y
D01F  20  A2  D0                JER   CSHFT2 (SHAPEL,H) WILL. SPECIFY
D022  C8                        INY   32 SEPARATE PAGES.
D023  D0  F6                    BNF   BKGND1 THROUGHOUT THE INIT.
D025  E6  18                    INC   SHAPEH
D027  A5  lB                    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-REQ
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
D030  4A                        LSR   ;IN HBASL, HBASH FOR
D03E  05  26                    ORA   HBASL      ACCESSING SCREEN MEN
D040  85  24                    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  PA                        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                    BEG   HPOSN2 (QUOTIENT) AND BIT
D05E  A0  23                    LDY   #$23 WITHIN SCREEN MEM BYTE
D060  69  04                    AOC   ##4 (MASK SPEC'D BY REMAINDER)
D062  C8              HPOSN1    INY
D063  E9  07          HPOSN2    SBC   #$7 SUBTRACT OUT SEVENS.
D065  80  F8                    BCS   HPOSN1
D067  8C  25  03                STY   MNDX WORKS FOR XO FROM
D06A  AA                        TAX   0 TO 279, LOW-ORDER
D06B  BD  EA  D0                LDA   MSKTBL-249, X BYTE IN X-REQ
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 FDR 180 DEGREE SHIFT
D077  80  29                    BCS   CSHFT2 PRIOR TO COPYING TO HCOLOR1.
D079.60                         RTS
D07A  20  2E  D0      HPLOT     JSR   HPOSN
D07D  A5  IC          HPLOT1    LOA   HCOLOR1 CALC BIT POSN IN HBASL,H
D07F  51  24                    EOR   (H8ASL),Y HNDX AND HMASK FROM
D031  25  30                    AND   HMASK Y-COORD IN A-REQ.
D033  51  26                    EOR   (HBASL), Y X-COORD IN X,Y-REGS.
D035  91  26                    STA   (HBASL),Y FOR ANY 'L' BITS OF HMAS
D037  60                        RTS   SUBSTITUTE CORRESPONDING
                                                 BIT OF HCOLOR1.
```

```
                         140   HI-RES GRAPHICS L, R, U, D SUBRS
D038  10  24             141   LFTRT    BPL   RIGHT USE SIGN FOR LFT/RT SELECT
D03A  A5  30             142   LEFT     LDA   HMASK
D03C  4A                 143            LSR   SHIFT LOW-ORDER
D03D  B0  05             144            BCS   LEFT1 7 BITS OF HMASK
D03F  49  C0             145            EOR   #$C0 ONE BIT TO LSB
DO91  85  30             146   LRI      STA   HMASK
D093  60                 147            RTS
D094  88                 148   LEFT1    DEY   DECR HORIZ INDEX.
D095  10  02             149            SPL    LEFT2
D097  A0  27             150            LDY   #$27 WRAP AROUND SCREEN
D099  A9  C0             151            LOA    #$C0 NEW HMASK, RIGHTMOST
D09B  85  30             152            STA   HMASK DOT OF BYTE
D0BD  SC  25  03         153            STY   HNDX UPDATE HORIZ INDEX
D0A0  A5  IC             154   CSHIFT   LDA   HCOLOR1
D0A2  0A                 155   CSHFT2   ASL   ; ROTATE LOW-ORDER
D0A3  C9  C0             156            CMP   #$C0 7 BITS OF HCOLDR1
D0A5  10  06             157            SPL   RTSI ONE BIT POSN.
D0A7  A5  1C             158            LDA   HCOLOR1
D0A9  49  7F             159            EOR   #$7F ZXYXYXYX —> ZYXYXYXY
D048  85  IC             160            STA   HCOLOR1
D0AD  60                 161            RTS
D0AE  A5  30             162            LDA   HMASK
D090  0A                 163            ASL   ; SHIFT LOW—ORDER
D0B1  49  80             164            EOR   #$80 7 BITS OF HMASK
D0B3  30  DC             165            BMI   LR1 ONE SIT TO MSB.
D0B5  A9  81             166            LDA   #$81
D0B7  C8                 167            INY   NEXT BYTE.
D0B8  C0  28             168            OPY   #$28
D0B4  90  DF             169            BCC   NEWNDX
D0BC  A0  00             170            LDY   #$0 WRAP AROUND SCREEN IF > 279
D0BE  B0  DB             171            BCS   NEWNDX ALWAYS TAKEN.

                         173   *L,R,U,D,      SUBROUTINES.
D0C0  18                 174   LRUDXI   CLC   NO 90 DEG ROT (X-0R).
D0C1  A5  51             175   LRUDX2   LOA   SHAPEX
D0C3  29  04             176            AND   #$4 IF B2=0 THEN NO PLOT.
D0C5  F0  27             177            BEG   LRUD4
D0C7  A9  7F             178            LDA   #$7F FOR EX-OR INTO SCREEN MEM
D009  25  30             179            AND   HMASK
D0CB  31  26             180            AND   (HSASL),Y          SCREEN BIT SET?
D0CD  D0  1B             181            BNE   LRUD3
D0CF  SE  24  03         182            INC   COLLSN
D0D2  A9  7F             183            LDA   #$7F
D0D4  25  30             184            AND   HMASK
D0D6  10  12             185            BPL   LRUD3 ALWAYS TAKEN.
D0D8  18                 186   LRUD1    CLC   NO 90 DEG ROT.
D0D9  AS  51             187   LRUD2    LDA   SHAPEX
D0DB  29  04             188            AND   #$4 IF B2=0 TNSN NO PLOT.
D0DD  F0  0F             189            BEQ   LRUD4
D0DF  B1  26             190            LDA   (HBASL), V
D0EI  45  1C             191            EOR   HCOLOR1 SET HI-RES SCREEN BIT
D0E3  25 `30             192            AND   HMASK TO CORRESPONDING HCOLOR1

D0E5  D0  03             193            BNE   LRUD3 IF BIT OF SCREEN CHANGES
D0E7  EE  2A  03         194            INC   COLLSN THEN INCR COLLSN DETECT
D0ZA  51  26             195   LRUD3    EOR   (HBASL), Y
D0EC  91  26             196            STA   (HBASL). Y
D0ZE  A5  51             197   LRUD4    LDA   SHAPEX ADD QDRNT TO
D0F0  65  53             198            ADC   QDRNT SPECIFIED VECTOR
D0F2  29  03             199            AND   #$3 AND MOVE LFT, RT,
                         200   EQS      EQU   *-I UP, OR DWN BASED
D0F4  09  02             201            CMP   #$2 ON SIGN AND CARRY.
D0F6  6A                 202            ROR
D0F7  B0  8F             203   LRUD     BCS   LFTRT
D0F9  30  30             204   UPOWN    BM1   DOWN4 SIGN FOR UP/DWN SELECT
D0F8  18                 205   UP       CLC
D0FC  A5  27             204            LDA   HBASH CALC BASE ADDRESS
D0FE  2C  EA  01         207            BI1   EQ1C (ADR OF LEFTMOST BYTE)
D101  00  22             208            ENS   UP4 FOR NEXT LINE UP
D103  06  24             209            ASL   HBASL 1N (HEASL, HBASH)
```

```
D105  B0  1A                        DCS     UP2-WITH 192-LINE WRAPAROUND
D107  2C  F3  D0                    BIT     EG3
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      #$23
D113  48                            PHA
D114  A5  24                        LDA.    HBASL HDASL = EABAB000
D116  69  B0                        ADC     #$B0 HBASM = 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     #$1F
D123  66  26           UP3          ROR     HBASL
D125  69  FC           UP4          AOC     #$FC
D127  85  27           UPDWN1       STA     HBASH
D129  60                            RT8
D12A  18               DOWN         CLC
D12B  A5  27           DOWN4        LDA     HBASH
D12D  69  04                        ADC     #$4 CALC BASE ADR FOR NEXT LINE
                       E04          EOU     *-1 DOWN TO (HBASL,HBASH)
D12F  2C  EA  D1                    BIT     EGIC
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     EG4
D13E  F0  13                        BEG     DOWN2
D140  A5  26                        LDA     HBASL
D142  69  50                        ADC     #$50
D144  49  F0                        EOR     #$F0
D146  F0  02                        BEG     DOWN3
D148  49  F0                        EOR     #$F0
D14A  85  26           DOWN3        STA     HBASL
D14C  AD  26  03                    LDA     HPAG
D14F  90  02                        BCC     00WN2
D151  69  E0           DOWN1        ADC     #$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 (X0L X0H) AND
D15A  8D  20  03                    STA     X0L Y0 TO ZERO FOR
D15D  8D  21  03                    STA     X0H REL LINE DRAW
D160  8D  22  03                    STA     Y0 (DX, DY).
D163  68                            PLA
D164  48               HLIN         PHA     ON ENTRY
D165  38                            SEC     XL: A-REG
D166  ED  20  03                    SBC     X0L XH; X-REG
D169  48                            PHA     Y: Y-REQ
D16A  8A                            TXA
D16B  ED  21  03                    SBC     X0H
D16E  85  53                        STA     QDRNT CALC ABS(X-X0)
D170  B0  0A                        BCS     HLIN2 IN (DXL.DXH)
```

```
D172   68                            PLA
D173   49  FE                        EOR    #$FF X DIR TO SIGN BIT
D175   69  01                        ADC    #$1 OF QDRNT.
D177   48                            PHA    0=RIGHT (DX P0S)
D178   A9  00                        LDA    #$0 1=LEFT (DX NEC)
D17A   E5  53                        SBC    QDRNT
D17C   85  51              HL1N2     STA    DXH
D17E   85  55                        STA    EH INIT (EL,EH) T0
D180   68                            PLA    ARS(X-X0)
D181.  85  50                        STA    DXL
D183   85  54                        STA    EL
D185   68                            PLA
D186   8D  20  03                    STA    X0L
D189   8E  21  03                    STX    X0H
D18C   98                            TYA
D18D   18                            CLC
D18E   ED  22  03                    SBC    Y0 CALC -ABS(Y-0)-I
D191   90  04                        BCC    HL1N3 IN DY.
D193   49  FF                        EOR    #$FF
D195   69  FE                        ADO    #$FE
D197   85  52              HLIN3     STA    DY ROTATE Y DIR INTO
D199   BC  22  03                    STY    V0 GDRNT SIGN BIT
D19C   66  53                        ROR    QDRNT (0=UP. l=DOWN)
D19E   38                            SEC
D19F   E5  50                        SBC    DXL INIT (COUNTL, COUNTH).
D1A1   AA                            TAX    TO -(DELTX+DELTY+1)
D1A2   A9  FE                        LDA    #$FF
D1A4   ES  51                        SBC    DXH
D1A6   65  1D                        STA    COUNTH
D1A8   AC  25  03                    LDY    HNDX HORIZ INDEX
D1AB   80  05                        BCS    MOVEX2 ALWAYS TAKEN.
D1AD   0A                  MOVEX     ASL    ; MOVE IN X-DIR. USE
D1AE   20                            JSR    LFTRT QDRNT 86 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   65  54                        STA    EL NOTE: DY IS (-DELTY)-1
D1B8   AS  55                        LDA    EH CARRY CLR IF (EL,EX)
D1BA   E9  00                        SBC    #$0 GOES NEG
D1BC   85  S5              HCOUNT    STA    EH
D1BE   81  26                        LDA    (HRASL).Y SCREEN BYTE.
D1C0   45  1C                        EOR    HCOLOR1 PLOT DOT OF HCOLOR1.
D1C2   25  30                        AND    HMASK CURRENT BIT MASK.
D1C4   51  26                        EOR    (HRASL), Y
D1CS   91  26                        STA    (HSASL), Y
D1C8   E8                            INX    DONE (DELTX+DELTY)
D1C9   00  04                        BNE    XLIN4 DOTS?
D1CB   E6  10                        INC    COUNTH
D1CD   F0  68                        BEQ    RTS2 YES, RETURN.
D1CF   AS  53              HLIN4     LDA    GDRNT FOR DIRECTION TEST
D1D1   B0  DA                        BCS    MOVEX IF CLR SET. (EL, EH) POS
D1D3   20  F9  D0                    JSR    UPDWN IF CLR, NEG, MOVE YDIR
D1D6   18                            CLC
D1D7   AS  54                        LDA    EL (EL., EH)+DELTX
D1D9   65  50                        ADC    DXL TO (EL,EH).
D1DB   65  54                        STA    EL
D1D0   A5  55                        LDA    EH CAR SET IF (EL,EH) GOES P0S
D1D6   65  51                        ADC    DXH
D1E1   50  09                        BVC    HCOUNT ALWAYS TAKEN.
D1E3   81                  MSKTBL    HEX    81 LEFTMOST BIT OF BYTE
D1E4   82  84  89                    HEX    82, 84, 86
D1E7   90  A0                        HEX    90, A0
D1E9   C0                            HEX    C0 RIGNTMOST BIT OF BYTE.
D1EA   1C                  EG1C      HEX    IC
D1EB   FE  FE  FA          COS       HEX    FF, FE, FA, F4. EC, El, D4, DS, B4
D1F4   Al  8D  78                    HEX    A1,8D, 78,61,49,31, 18.FF
```

```
                        336  *  HI-RES GRAPHICS COORDINATE RESTORE SUSR
D1FC  A5  26            337  HFIND    LDA   HBASL
D1FE  0A                338           ASL   CONVERTS BASE ADR
D1FF  A5  27            339           LDA   HBASH TO Y-COORD.
D201  29  03            340           AND   #$3
D203  2A                341           ROL   ; FOR HBASL = EABASOOO
D204  05  26            342           ORA   HBASL HBASH = PPPFGHCD
D206  0A                343           ASL
D207  0A                344           ASL   ; GENERATE
D208  0A                345           ASL   ; Y-COORD = ABCDEFGH
D209  8D  22  03        346           STA   Y0
D20C  A5  27            347           LDA   HBASH (PPP.SCREEN PAGE,
D20E  4A                348           LSR   ; NORMALLY 001 FOR
D20F  4A                349           LSR   ; $2000-$3FFF
D210  29  07            350           AND   #$7 HI-RES SCREEN)
D212  0D  22  03        351           ORA   Y0
D215  8D  22  03        352           STA   Y0 CONVENTS HNDX (INDEX
D218  AD  25  03        353           LDA   HNDX FROM BASE ADR)
D21B  0A                354           ASL   ; AND HMASK (BIT
D21C  6D  25  03        355           ADC   HNDX MASK TO X-COORD
D21F  0A                356           ASL   ; IN (XOL,XOH)
D220  AA                357           TAX   (RANGE $0—$133)
D221  CA                358           DEX
D222  A5  30            359           LDA   HMASK
D224  29  7F            360           AND   #$7F
D226  E8                361  HFIND1   INX
D227  4A                362           LSR
D228  D0  FC            363           BNE   HFIND1
D22A  8D  21  03        364           STA   XOH
D22D  8A                365           TXA
D22E  18                366           CLC   CALC HNDX*7 +
D22F  6D  25  03        367           ADC   HNDX LOG (BASE 2) HMASK
D232  90  03            368           BCC   HFIND2
D234  EE  21  03        369           INC   X0H
D237  8D  20  03        370  HFIND2   STA   X0L
D23A  60                371  RTS2     RTS
                        373  *  HI-RES GRAPHICS SHAPE DRAW SUBR
                        374  *
                        375  * SHAPE  DRAW
                        376  *  R = 0 TO 63
                        377  * SCALE  FACTOR USED (1=NORMAL)
                        378  *
D23B  86  1A            379  DRAW     STX   SHAPEL DRAW DEFINITION
D23D  84  1B            380           STY   SHAPEH POINTER.
D23F  AA                381  DRAWl    TAX
D240  4A                392           LSR   ; ROT ($0-$3F)
D241  4A                383           LSR
D942  4A                384           LSR   ; QDRNT  0=UP, 1=RT.
D243  4A                385           LSR   ; 2=DWN, 3=LFT.
D244  85  53            386           STA   QDRNT
D246  8A                387           TXA
D247  29  0F            389           AND   #$F
D249  AA                389           TAX
D24A  BC  EB  Dl        390           LDY   COS, X SAVE COS AND SIN
D24D  84  50            391           STY   DXL VALS IN DXL AND DY
D24F  49  0F            392           E0R
D251  AA                393           TAX
D252  BC  EB  Dl        394           LDY   CDS+1.X
D255  C8                395           I NY
D256  84  52            396           STY   DY
D258  AC  25  03        397  DRAW2    LDY   HNDX BYTE INDEX FROM
D25B  A2  00            398           LDX   #$0 HI-RES BASE ADR.
D25D  8E  2A  03        399           STX   COLLSN CLEAR COLLISION COUNT,
D260  A1  1A            400           LDA   (SHAPEL,X) 1ST SHAPE DEF BYTE.
```

```
D262   85 51          DRAWS    STA   SHAPEX
D264   A2 80                   LDX   #$80
D266   86 54                   STX   El, EL, EH FOR FRACTIONAL
D268   86 55                   STX   EH L,R,IJ,D VECTORS.
D26A   AE 27 03                LDX   SCALE SCALE FACTOR.
D26D   A5 54          DRAW4    LDA   EL
D26F   38                      SEC   IF FRAC COS 0VFL
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                   SCC   DRAW6 DIRECTION +90 DEG.
D282   20 09 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   00 03                   BNE   DRAW3 NOT DONE THIS BYTE.
D28F   E6 lA                   INC   SNAPEL
D291   00 02                   BNE   DRAW3 NEXT BYTE OF
D293   56 12                   INC   SHAPEH SHAPE DEFINITION.
D295   Al 1A          DRAW7    LDA   (SHAPEL, X)
D297   D0 C9                   BNE   DRAW3 DONE IF ZERO.
D299   60                      RTS

                      *   HI-RES GRAPHICS SHAPE EX-OR SUBR
                      *
                      *   EX-0R SHAPE INTO SCREEN.
                      *
                      *   ROT = 0 TO 3 (QUADRANT ONLY)
                      *   SCALE IS USED
                      *
D29A   86 1A          XDRAW    STX   SHAPEL SHAPE DEFINITION
D29C   84 15                   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 E8 D1                LDY   COS. X SAVE COS AND SIN
D2AC   84 50                   STY   DXL VALS IN DXL AND DY,
D24E   49 0F                   EOR   #$F
D280   AA                      TAX
D2B1   SC EC D1                LDY   COS+1, X
D2B4   C8                      I NY
D2B5   84 52                   STY   DY
D2B7   AC 25 03       XDRAW2   LDY   HNDX INDEX FROM HI-RES
D2BA   A2 00                   LDX   #$0 BADE ADR.
D2BC   8E 2A 03                STX   COLLSN CLEAR COLLISION DETECT
D2BF   A1 1A                   LDA   (SHAPEL,X) 1ST SHAPE DEF BYTE.
```

```
D2C1  05  51       XDRAW3  STA   SHAPEX
D2C3  A2  80               LDX   #$80
D2C5  96  54               STX   EC 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               LDA   El
D2CE  38                   SEC   IF FRAC COS OVFL
D2CF  65  50               A0C   DXL THEN MOVE IN
D2D1  85  54               STA   EL SPECIFIED VECTOR
D2D3  90  04               BCC   XDRAWS DIRECTION
D2D5  20  C0  D0           JSR   LRUDX 1
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 DEC.
D2E1  20  D9  DO           JSR   LRIJD2
D2E4  CA           X0RAW6  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  DO  03               BNE   XDRAW3
D2EE  E6  IA               INC   SHAPEL
D2F0  D0  02               BNE   XDRAW7 NEXT BYTE OF
D2F2  E6  1B               INC   SHAPEH SHAPE DEF.
D2F4  Al  1A       XDARW7  LDA   (SHAPEL, X)
D2F6  DO  C9               BNE   XDRAW3 DONE IF ZERO.
D2F8  60                   RTS

                   .*  ENTRY POINTS FROM APPLE—Il BASIC
D2F9  20  90  D3   BPOSN   JSR   PCOLR POSN CALL COLR FROM BASIC
D2FC  8D  24  03           STA   HCOLOR
D2FF  20  AF  D3           JSR   GETY0 Y0 FROM 8ASIC.
D302  48                   PHA
D303  20  9A  D3           JSR   GETX0 X0 FROM BASIC.
D306  68                   PLA
D307  20  2E  D0           JSR   HPOSN
D30A  AE  23  03           LDX   BXSAV
D30D  60                   RTS
D30E  20  F9  02   BPLOT   JMP   BPOSN PLOT CALL (BASIC).
D311  4C  7D  D0           JMP   HPL0T1
D314  AD  25  03   BLIN1   LDA   HNDX
D317  4A                   LSR   ; SET HCOLORI FROM
D318  20  90  D3           JSR   PCOLR BASIC VAR COLR.
D31B  20  75  D0           JSR   HPOSN3
D31E  20  9A  03   BLINE   JSR   GETX0 LINE CALL, GET X0 FROM BASIC
D321  8A                   TXA
D322  48                   PHA
D323  98                   TYA
D324  AA                   TAX
D325  20  AF  D3           JSR   GETY0 Y0 FROM BASIC
D328  A8                   TAY
D329  68                   PLA
D32A  20  64  D1           JSR   HL IN
D32D  AE  23  03           LDX   BXSAV
D330  60                   RTS
D331  20  90  D3   BGND    JSR   PCOLR BACKGROUND CALL
D334  4C  10  D0           JMP   BKGNDO
```

```
                   * DRAW ROUTINES
D337  20  F9  D2   DORAWI .   JSR    BPOSN
D33A  20  51  D3   BDRAW      TSR    BDRAWX DRAW CALL FROM BASIC.
D33D  20  3B  D2              JSR    DRAW
D340  AE  23  D3              LDX    DXSAV
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 FROM BASIC
D354  AO  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 PROM BASIC.
D361  48                      PHA    SAVE ON STACK.
D362  AD  28  03              LDA    SHAPEXL
D365  85  lA                  STA    SHAPEL START OF
D367  AD  29  03              LDA    SHAPXH SHAPE TABLE.
D36A  85  18                  STA    SHAPEH
D36C  AG  20                  LDY    #$20
D36E  20  92  03              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                      ASL
D37C  90  03                  BCC    BDRWX2
D37E  E6  1B                  INC    SNAPEH
D380  18                      CLC
D381  AB           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  60  29  03              ADC    SHAPXH (X LOW. Y HI)
D38B  A8                      TAY
D38E  68                      PLA    ROT FROM STACK.
D38F  60                      RTS

D390  A0  16       * BASIC PARAM FETCH SUBR'S
D392  B1  4A       PCOLR      LDY    #$16
D394  D0  16       PBYTE      LDA    (LOMEML), Y
D396  88                      BNE    RERRI GET BASIC PARAM.
D397  Bl  4A                  DEY    (ERR IF >255)
D399  60                      LDA    (LOMEML). V
D39A  8E  23  03   RTSB.      RTS
D39D  A0  05       GETYO      STX    BXSAV SAVE FOR BASIC.
D39F  B1  4A                  LDY    #$5
D3A1  AA                      LDA    (LOMEML),Y X0 LOW-ORDER BYTE.
D3A2  C8                      TAX
D3A3  B1  4A                  INY    .
D3A5  A8                      LDA    (LOMEML),Y HI—ORDER BYTE
D3A6  E0  18                  TAY
D3A8  E9  01                  CPX    #$18
D3AA  90  ED                  SBC    #$1 RANGE ERR IF >279
D34C  4C  68  EE              BCC    RTSB
D3AF  A0  0D       RERR1      JMP    RNGERR
D3BI  20  92  D3   GETYO      LDY    #$D OFFSET TO Y0 FROM LOMEM
D3B4  C9  C0                  JSR    PBYTE GET BASIC PARAM YO
D3B6  80  F4                  CMP    #$C0 (ERR IF >191)
D3B8  60                      BCS    RERR1
                              RTS
```

```
                    590   *SHAPE TAPE LOAD SUSROUTINE
D3B9  8E  23 03     591  SHLOAD  STX   SXSAV SAVE FOR SASIC.
D3BC  20  1E F1     592          JSR   ACAOR READ 2-BYTE LENGTH INTO
D3BF  20  FD FE     593          JSR   READ BASIC ACC
D3C2  A9  00        594          LDA   #$00 START OF SHAPE TABLE IS $0800
D3C4  85  3C        595          STA   A1L
D3C6  8D  28 03     596          STA   SHAPXL
D3C9  18            597          CLC
D3CA  65  CE        598          ADC   ACL
D3CC  A8            599          TAY
D3CD  A9  08        600          LDA   #$08 HIGH BYTE OF SHAPE TABLE POINTER
D3CF  85  3D        601          STA   A1H
D3D1  8D  29 03     602          STA   SHAPXL
D3D4  65  CF        603          ADC   ACH
D3D6  B0  25        604          BCS   MFULL1 NOT ENOUGH MEMORY.
D3D8  C4  CA        605          CPY   PPL
D3DA  48            606          PHA
D3DB  E5  CE        607          SEC   PPH
D3DD  68            608          PLA
D3DE  B0  1D        609          BCS   MFULL1
D3E0  54  3E        610          STY   A2L
D3E2  55  SF        611          STA   A2H
D3E4  CS            612          INY
D3E5  00  02        612          BNE   SHLOD1
D3E7  69  01        614          ADC   #$1
D3E9  54  4A        615  SHLOD1  STY   LOMEML
D3EB  55  4B        616          STA   LOMENH
D3ED  54  CC        617          STY   PVL
D3EF  55  CD        618          STA   PVH
D3F1  20  FA FC     619          JSR   RD2BIT
D3F4  A9  03        620          LDA   #$3 . 5 SECOND HEADER
D3F6  20  02 FF     621          JSR   READX1
D3F9  AE  23 03     622          LDX   BXSAY
D3FC  60            623          RTS
D3FD  4C  6B E3     624  MFULL1  JMP   MEM FUL

- - - END ASSEMBLY - - -

TOTAL ERRORS: 00
```

```
****************************************************
*                                                  *
* APPLE-][ BASIC RENUMBER/ APPEND SUBROUTINES      *
*                                                  *
*                 VERSION    TWO                   *
*                   RENUMBER                       *
*                  >CLR                            *
*                  >START=                         *
*                  >STEP=                          *
*                  >CALL-10531                     *
*                                                  *
*                   OPTIONAL                       *
*                  >FROM=                          *
*                  >T0=                            *
*                  >CALL -10521                    *
*                                                  *
*                  USE RENX ENTRY                  *
*                 FOR  RENUMEER  ALL               *
*                                                  *
*            WOZ         APRIL  12,   1978         *
*            APPLE   COMPUTER   INC.               *
****************************************************

*
*           6502 EQUATES
*
ROL         EQU     $0      LOW-OROER SW16 RO 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
MEMEULL     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   SWEET 16 ENTRY
CROUT       EQU     $FD8E   CAR RET SUBR.
COUT        EQU     $FDED   CHAR OUT SUBR.

*
*             SWEET 16 EQUATES
*
ACC.        EQU     $0      SWEET 16 ACCUMULATOR.
NEULOW      EQU     $1      NEW INITIAL LNO.
NEWINCR     EQU     $2      NEW LNO INOR.
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.
SCRB        EQU     $8      SCRATCH REG.
HMEM        EQU     $8      HIMEM (END OF PRGM).
SCR9        EQU     $9      SCRATCH REQ.
PRGNDX      EQU     $9      PASS 1 PROC INDEX.
PRONDXI     EQU     $A       ALSO PROC INDEX,
NEWLN       EQU     $B      NEXT “NEW UND”.
NEWLN1      EQU     $C      PRIOR “NEW LNO” ASSIGN.
TBLND       EQU     $6      PASS 2 LNO TABLE END,
PRGNDX2     EQU     $7       PASS 2 PROG INDEX.
CHRO        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 F03 UPDATE.
                   STRCON     EQU    $B           BASIC STR CON TOKEN.
                   REM        EQU    $C           BASIC REM TOKEN.
                   R13        EQU    $D           SWEET 16 REG 13 (CPR NEC).
                   THEN       EQU    $D           BASIC THEN TOKEN
                   LIST       EQU    $D           BASIC LIST TOKEN
                   DEL        EQU    $D
                   SCRC       EQU    $C           SCRATCH REQ 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     LNH I
D406   F4                     DCR    LNH I
D407   00                     RTN
D408   20  39  F6  RENUM      JSR    SW16
D40B  18  4C  00              SET    HMEM, HIMEM
D40E   68                     LDD    @HMEM
D40W        38                ST     HMEM
D410   19  CE  00  RNUM3      SET    SCR9, PVL+2
D413   C9.                    POP D  @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 TOLIM AND PRGNDX.
D41B  39                      ST     PRGNDX
D41C  29                      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 ‘lUMBER.
D42A  D3                      CPR    LNLOW         IF < LNLOW THEN OOTO 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   46  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    C OUT
D445   A5  17                 LDA    R11H
D447   A6  16                 LDX    R11L
D449   20  1B   E5            JSR    PRDEC
D44C  20   8E  FD             JSR    CROUT
                   *
D44F 20 BC F6                 JSR    SW16+3       **** END 6502 CODE ****
```

```
                          *
D452  2B                          LD    NEWLN
D453  3C                          ST    NEWLNI     COPY NEWLN ro NEWLMI AND INCR
D454  A2                          ADD   NEWINCR  UEWLN flY NEWINOR
D455  3B                          ST    NEWLN
D456  0D                          HEX   .00           'NUL' (WELL SKIP NEXT INSTRUCTION)
D457  D1                  P1B     CPR   NEWLOW .IF LOW LNO< NEW LOW THEN RANGE ERR
D45B  02 C2                       BNC   PASS1
D45A  00                  RERR    RTN   PRINT "RANGE ERR" MESSAGE AND RETURN.
D450  4C 68 EE                    JMP   RANGERR
D45E  00                  KERR    RTN   PRINT "MEM FULL" MESSAGE AND RETURN
D45F  4C 6B E3                    JMP   MEMFULL
D462  EC                  PlC     INR   NEWLN1     IF HI LNO <= MOST RECENT HEWLN THEN
D463  DC                          CRR   NEWLN1              RANGE ERROR.
D464  02   F4                     BNC   RERR

                          *
                          *          APPLE ][ BASIC RENUMBER / APPEND SUBROUTINE -- PASS 2
                          *
D466  19 B0 00            PASS2   SET   CHRO, $00B0       ASCII "0"
D469  1A CO 00                    SET   CHRA, $00C0       ASCII "A"
D46C  27                  P2A     LD    PRGNDX2
D46D  D8                          CPR   HMEM       IP PROG INDEX = HIMEN THEN DONE PASS 2.
D46E  03   63                     BC    DONE
D470  E7                          INR   PRONDX2 SKIP LENIN BYTE
D471  67                          LDD   @PRGNDX2 LINE NUMBER
D472  3D                  UPDATE  ST    OLDLN      SAVE OLD LUD.
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                          AD0   NEWINCR ADD INCR TO NEWLN1.
D479  3C                          ST    NFWLN1
D47A  28                          LD    TBLNDX2    IF LNO TBL lDX = TBLND THEN DONE
D47B  B6                          SUB   TELND      SCANNING LNO TABLE
D47C  03   07                     BC    UD3
D47E  6B                          LDD   @TBLNDX2NEXT LNO FROM TABLE.
D47F  8D                          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  2800            UD3     SET   STRCON, #$028            STR CON TOKEN.
D488  1C                          HEX   IC         (SKIP-S NEXT TWO INSTRUCTIONS)
D489  67                  GOTCON  LDD   @PRGNDX2
D48A  FC                          DCR   MODE       IF MODE = 0 THEN UPDATE LNO REF.
D48B  08   E5                     BM1   UPDATE
D48C  47                  ITEM    LD    @PRGNDXBASIC TOKEN.2
D48E  D9                          CPR   CHRO
D4SF  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.
D496 F7                           DCR   PRGNDX2
D499 47                           LD    @PRGNDX2
```

```
D49A  DB          191  CHKTOK  CPR    STRCON    SW CON TOKEN?
D49B  06   F7     192          BZ     SKPASC    YES, SKIP SUBSEOUENT BYTES.
D49D  1C   5D  00 193          SET    REM, $0050
D4A0  DC          194          CPR    REM       REM TOKEN?
D4A1  06F1        195          BZ     SKPASC    YES, SKIP SUBSEQUENT LINE
D4A3  08   13     196          BM1    CONTST    GOSUB, LOOK FOR LINE NUMBER.
D4A5  FD          197          DCR    R13
D4A6  FD          198          DCR    R13       (TOKEN $5F IS GOTO)
D4A7  06   0F     199          BZ     CONTST
D4A9  10   24  00 200          SET    THEN, $0024
04AC  DD          201          CR9    THEN
D4AD06   09       202          BZ     CONTST    'THEN' LNO, LOOK FOR LNO.
D4AF  F0          203          DCR    ACC
D4B0  06   116    204          BZ     P2A       E0L (TOKEN 01)?
D4B2  10   74  00 205          SET    LIST, $0074
D4B5  20          206          SUB    LIST      SET MODEIF LIST OR LIST COMMA.
D4B6  09   01     207          BNM1   CONTS2    (TOKENS $74, $75)
D4B8  20          208  CONTST  SUB    ACC       CLEAR MODE FOR LNO
D4B9  3C          209  CONTS2  ST     MODE      UPDATE CHECK
D4BA01   01       210          BR     ITEM

                  212 *
                  213 *        .
                  214 *        APPLE ][ BASIC APPEND SUBROUTINE
                  215 *
D4BC20   89  F6   216  APPEND  JSR    SW1 6
D4BF  1C   4E  00 217          SET    SCRC, HIMEM+2
D4C2  CC          218          POPD   @SCRC     SAVE HIMEM.
D4C0  88          219          ST     HMEM
D4C4  19   CA00   220          SET    SCR9, PPL
D4C7  69          221          L0D    @SCR9
D4C8  7C          222          ST D   @SCRC     SET HIMEM TO PRESERVE PROGRAM.
D4C9  00          220          RTN
D4CA20   DF  F0   224          JSR    LOAD      LOAD FROM TAPE
D4CD20   89  F6   225          JSR    SW16
D4.00CC           226          P0PD   @SCRC     RESTORE HIMEM TO SHOW BOTH PROGRAMS
D4Ot  28          227          LD     HMEM      (OLD AND NEW)
D402  7C          228          STD    RETURN
D403  00          229 DONE     RTN
D404  60          230          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
*
ROL.          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 DISASSEMILER 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 '4'?
D4E3  D0  0C                    BNE  RELOC2 NO, RELOC CODE SEQ.
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=
D4FO  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                    BEG  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  99  F6    XLATE       JSR  SW16 IF ADR OF ZERO PAGE
D512  22                        LD   FRMEND OR ABS IS IN SOURCE
D513  06                        CPR  ADR (FRM) BLOCK THEN
D514  02  06                    BNC  SW16RT SUBSTITUTE
D516  26                        LD   ADR ADR-SOURCE SEG+DEST BEG
D517  B1                        SUB  FRMBEG
D518  02  02                    BNC  SW16RT
D5lA  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  06  2F                    DEC  LENGTH DEST SEGMENT. UPDATE
D529  10  F4                    BPL  STINS2 SOURCE, DEST SEGMENT
D52B  90  C4                    DCC  RELOC2 POINTERS. LOOP IF NOT
D52D  60                        RTS  BEYOND SOURCE SEQ END.
                                          END ASSEMBLY

- - -END ASSEMBLY - - -

TOlAL ERRORS: 00
```

```
                     ********************************
                     *                              *
                     *         TAPE VERIFY          *
                     *                              *
                     *           JAN 78             *
                     *           BY WOZ             *
                     *                              *
                     *                              *
                     ********************************
                     *
                     *         TAPE VERIFY EGUATES
                     *
                      CHKSOM 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   $F092 ;PRINT (A1)-
                     PRBYTE  EQU   $FDDA
                     COUT    EQU   $FDED
                     FINISH  EQU   $FF26 ;CHECK CHECKSUM,  RING SELL
                     PRERR   EQU   $FF2D
                     *
                     *          TAPE VERIFY ROUTINE
                     *
                             0RG   $D535
                             0BJ   $A535
                     VFYBSC  STX   XSAVE ;PRESERVE X-REG FOR BASIC
0535  86  D8                 SEC
0537  38                     LDX   #$FF
0538  A2  FF         GET LEN LDA   HIMEM+1 CALCULATE PROGRAM LENGTH
053A  A5  4D                 SBC   PP+1,X INTO PRLEN
053C  FS  CB                 STA   PRLEN+1, X
D53E  95  CF                 INX
0540  E8                     BEQ   GETLEN
0541  F0  F7                 JSR   HDRSET ;SET UP POINTERS
0543  20  1E  F1             JSR   TAPEVFY ;DO A VERIFY ON HEADER
0546  20  54  D5             LDX   #$01 ;PREPARE FOR PRGSET
0549  A2  01                 JSR   PRGSET SET POINTERS FOR PROGRAM VERIFY
054B  20  2C  F1             JSR   TAPEVFY
DS4E  20  54  D5             LDX   XSAVE ;RESTORE X-REG
0551  A6  D8                 RTS
0553  60
```

```
                  *
                  * TAPE VERIPY RAM IMAGE (A1.A2)
                  *
D554  20 FA FC    TAPEVPY  JBR  RD2BIT
D557  A9 16                LDA  #$16
D559  20 C9 FC             JSR  HEADR ;SYNCHRONIZE ON HEADER
D55C  85 2E                STA  CHKSUM INITIALIZE CHKSUM
DS5E  20 FA FC             JSR  RD2B1T
D561  A0 24       VRPY2    LDY  #$24
D563  20 PD PC             JSR  RDBIT
D566  20 P9                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                SEQ  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  #$34 ONE LESS THAN USED IN READ FOR EXTRA 12
D57B  90 F0                BCC  VRPY3 ;LOOP UNTIL A1>A2
D57D  4C 26 FF             JMP  FINISH ;VERIPY CHECKSUMSCRING BELL
D580  EA          EXTDEL   NOP  ;EXTRA DELAY TO EQUALIZE TIMING
D581  EA                   NOP  ;    ( +12 USEC )
D582  EA                   NOP
D583  C1 3C                CMP  (AI,X) BYTE THE SAME?
D585  F0 EB                BEQ  VFYLOOP IT MATCHES, LOOP BACK
D587  48                   PHA  ;SAVE WRONG BYTE FROM TAPE
D598  20 2D FF             JSR  PRERR ;PRINT “ERR”
D58B  20 92 FD             JSR  PRA1 ;OUTPUT (A1)"-"
D58E  B1 3C                LDA  (A1),Y
D590  20 DA FD             JSR  PRBYTE OUTPUT CONTENTS OP A1
D593  A9 A0                LDA  #$A0 PRINT A BLANK
D595  20 ED FD             JSR  COUT
D598  A9 A8                LDA  #$AB ; ‘(’
D59A  20 ED FD             JSR  COUT
D59D  68                   PLA  ;OUTPUT BAD BYTE FROM TAPE
D59E  20 DA FD             JSR  PRBYTE
D5A1  A9 A9                LDA  #$A9 ;   ‘(’
D5A3  20 ED FD             JSR  COUT
D5A6  A9 FD                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   $D
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 SUSR.
COOT      EQU   $FDED   Cl-fAR OUT SUEBR
PRERR     EQU   $FF2D  PRINTS 'ERR - BELL'
BELL      EQU   $FF3A
```

```
                      •
                      *       RAMTEST
                      *
                              ORG     $D5BC
                              OBJ     $A5BC
D5BC  A9  C3          SETUP   LDA     #$C3 ;SET UP CNTRL-V LOCATION
D5BE  A0  D5                  LDY     #$D5
D5CO  4C  B0  D5              JMP     SETCTLY
D5C3  A9  00          RAMTST  LDA     #$0 TEST FOR $00.
DSC5  20  D0  05              JSR     TEST
D508  A9  FF                  LDA     #$FF THEN $FF.
D5CA  20  D0  D5              JSR     TEST
D500  4C  3A  FF              JMP     BELL
D500  85  00          TEST    STA     DATA
D502  49  FF                  E0R     #$FF
D504  85  01                  STA     NDATA
D506  A5  3D                  LDA     A1H
D508  85  07                  STA     R3H INIT (R3L, R3H)
DSDA  85  09                  STA     R4H   (R4L, R4H),  (R5L, R5H)
D500  85  0B                  STA     A4H   TO TEST BLOCK BEGIN
D50E  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).
D5EB  A5  00                  LDA     DATA
D5EA  91  08          TEST0I  STA     (R4L), Y SET ENTIRE TEST
D5EC  C8                      INY     BLOCK TO DATA.
D5ED  D0  FB                  BNE     TEST01
D5EF  E6  09                  INC     R4H
D5FI  CA                      DEX
D5F2  D0  F6                  BNE     TEST01
D5F4  A6  3E                  LDX     A2L
D5F6  B1  06          TEST02  LDA     (R3L).Y VERIFY ENTIRE
D5P9  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,
D606  20  8A  D6              JSR     PRBYSP
D606  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
D618  91  0A                  STA     (R5L),Y SET TEST CELL TO
D610  84  0D                  STY     R6H      NDATA AND R6
D61F  64  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.
D620  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                IPNC 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          RTSI      RTS
D645  85  02      TEST 6    STA  TESTD SAVE GALLOP DATA.
D647  A5  0A                LDA  R5L
D649  45  0C                EOR  R6L     SETR4 TO R5
D64B  85  08                STA  R4L     EX - OR R6
D64D  A5  0B                LDA  R5N     FOR NEIGHBOR
D64F  45  0D                EUR  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                BEG  RTSI  (OK).
D65D  48                    PHA  PRESERVE FAIL DATA.
D65E  A5  0B                LDA  R5N
D660  20  0A  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  #$8D ASCII CAR. RETURN.
D687  4C  ED  FD            JMP  COUT
D69A  20  DA  FD  PRBYSP    JSR  PRBYTE
D63D  A9  A0                LDA  #$A0 OUTPUT BYTE. THEN
D6SF  4C  ED  FD            JMP  COUT   SPACE.
                            ORG  $3F8
03F8  4C  C3  D5  USRLOC    JMP  RAMTST  ENTRY PROM MON (CTRL—Y)
```

- - - END ASSEMSLY - - -

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 SPEARER
D71F E6   02            PLAY2   INC    LENGTH ; DURATION
D721 D0   05                    BNE    PATH1 ; NOT EXPIPED
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                    BEG    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
D34   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
D7SF  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
```

```
                          62 *
                          63 * NOTE TASLE L00~SUP SUDROUTINE
                          64*
                          65* GIVEN NOTE NUMBER IN ‘NOTE’
                          66* DURATION COUNT IN ‘LONG’
                          67* FIND ‘UPTIME’ AND ‘DOWNTIME’
                          68*       ACCORDING TO DUTY CYCLE CALLED
                          69*       FOR BY ‘VOICE’
                          70*
D74E  AD  FF  02          71LOOKUP  LDA   NOTE GET NOTE NUMOER
D751  0A                  72        ASL   ; DOUBLE IT
D752  A8                  73        TAY
D753  B9  96  D7          74        LDA   NOTES, Y ; GET UPTIME
D756  85  00              75        STA   DOWNTIME ; SAVE IT
D758  AD  FD  02          76.       LDA   VOICE ; GET DUTY CYCLE
D752  4A                  77SHIFT   LSR
D75C  F0  04              78        BEQ   DONE ; SHIFT WIDTH COUNT
D75E  46  00              79        LSR   DOWNTIME ; ACCORDING TO VOICE
D760  D0  P9              90        BNE   SHIFT
D762  B9  96  D7          81DONE    LDA   NOTES, Y ; GET ORIGINAL
D765  38                  82        SEC
D766  E5  00              83        SBC   DOWNTIME ; COMPUTE DIFFERENCE
D768  85  01              84        STA   UPTIME ; SAVE IT
D76A  C8                  85        INY   ; NEXT ENTRY
D762  B9  96  D7          86        LDA   NOTES,Y ;  GET DOWNTIME
D76E  65  00              87        ADC   DOWNTIME ; ADD DIFFERENCE
D770  85  00              88        STA   DOWNTIME
D772  A9  00              89        LDA   #0
D774  38                  90        SEC
D775  ED  FE  02          91        SBC   LONG ; GET COMPLIMENT OF DURATION
D778  85  03              92.       STA   LENGTH+1 MOST SIGNIFICANT BYTE
D77A  A9  00              93        LDA   #0
D77C  85  02              94        STA   LENGTH.
D77E  A5  01              95        LDA   UPTIME
D780  D0  98              96        BNE   PLAY IF NOT NOTE #0, PLAY IT
                          97
                          98* ‘REST’ SUBROUTINE’ PLAYS NOTE #0
                          99*  SILENTLY, FOR SAME DURATION AS
                         100*  A REGULAR NOTE
D782  EA                 101*
D783  EA                 102REST    NOP   ; DUMMY
D784  4C  87  07         103        NOP   ; CYCLE USERS
D787  E6  02             104        JMP   REST2 ; TO ADJUST TIME
D789  D0  05             105REST2 . INC   LENGTH
D788  E6  03             106        BNE   REST3
D780  D0  05             107        INC   LENGTH+ 1
D78F  60                 108        BNE   REST4
D790  EA                 109        RTS   ; IF DURATION EXPIRED
D791  4C  94  D7         110RESTS   NOP   ; USE UP ‘INC’ CYCLES
D794  D0  EC             111        JMP   REST4
                         112REST4   BNE   REST ; ALWAYS TAKEN
```

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

 - - - END ASSEMBLY  - - -

TOTAL ERR0RS: 00
```

APPENDIX II
SUMMARY OF PROGRAMMER'S AID COMMANDS

Chapter 1: RENUMBER

(a) To renumber an entire BASIC program:

```
CLR
START  = 1000
STEP  = 10
CALL —10531
```

(b) To renumber a program portion:

```
CLR
START = 200
STEP = 20

FROM = 300        (program portion
TO = 500          to be renumbered)

CALL —10521
```

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 —11076
```

Chapter 3: TAPE VERIFY (BASIC)

(a) Save current BASIC program on tape:

```
SAVE
```

(b) Replay the tape, after:

```
CALL —10955
```

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 show 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

(4) 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 show 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*l00, must <u>not</u> be greater than starting address. One page = 256 bytes ($100 bytes, in Hex).

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

addr1.pagesl ctrl Y addr2.pages2 ctrl Y Addr3.pages3 ctrl Y return

Example, for a 48K system:

400.4 ctrl Y 800.8 ctrl Y 1000.10 ctrl Y 2000.20 ctrl Y
3000.20 ctrl Y 4000.40 ctrl Y 7000.20 ctrl Y 8000.40
ctrl Y return

(d) To repeat test indefinitely:

N complete test 34:0 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 = -10473
PITCH = 767
TIME = 766
TIMBRE = 765

(b) Set parameters for next note:

POKE PITCH, p	(p =1 to 50; 32 = middle C)
POKE TIME, m	(m = 1 to 255; 170 = 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 progratn):

```
1 X0 = Y0 = 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):

```
10   INIT   =  —12288    CLEAR =  —12274    BKGND = —11471
11   POSN  =  —11527    PLOT   =  —11506    LINE = —11500
12   DRAW =  —11465    DRAWl =  —11462
13   FIND   =  —11780    SHLOAD =—11335
```

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

```
20 BLACK = 0 : LET GREEN = 42 : VIOLET = 85
21 WHITE = 127 : ORANGE = 170 : BLIJE  213
22 BLACK2 = 128 : WHITE2 = 255
```

(d) Initialize:

```
30 CALL INIT
```

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

Example:

```
40 COLR = VIOLET : CALL BKCND : REM : TURN BACKGROUND VIOLET
50 FOR I = 0 TO 279 STEP 5
60 X0 = 140 : V0 = 150 : COLR = WHITE : REM SET PARAMETERS
70 CALL POSN : REM MARK THE ‘CENTER’
80 X0 = 1 : Y0 = 0 : REM SET NEW PARAMETERS
90 CALL LINE : REM DRAW LINE TO EDGE
100 NEXT I : END
```

QUICK REFERENCE TO HIGH-RESOLUTION INFORMATION

Subroutine Name	CALLing Address	Patameters Needed
INIT	—12288	
CLEAR	—12274	
BKGND	—11471	COLR
POSN	—11527	X0, Y0. COLR X0,
PLOT	—11506	Y0, COLR
LINE	—11300	X0, Y0, COLR
DRAW	—11463	X0, Y0, COLR, SHAPE, ROT. SCALE
DRAW1	—11462	SHAPE, ROT, SCALE
FIND	—11780	
SHLOAD	—11335	

Color Name	COLR Value	Color Name	COLR Value
BLACK	0	BLACK2	128
GREEN	42	ORANGE	170
VIOLET	85	BLUE	213
WHITE	127	WHITE2	255

(Note: on systems below S/N 6000. colors in the second column appear identical to those in the first column)

CHANGING THE High-Resolution GRAPHICS DISPLAY

Full—Screen Graphics	POKE —16302, 0
Mixed Graphics—Plus—Text (Default)	POKE —16301, 0
Page 2 Display	POKE —16299, 0
Page 1 Display (Normal)	POKE —16300, 0
Page 2 Plotting	POKE 806, 64
Page 1 Plotting (Default)	POKE 806, 32

(Note: CALL INIT sets mixed graphics—plus—text, and Page 1 plotting, but does not reset to Page 1 display.)

Collision Count for Shapes PEEK (810)

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