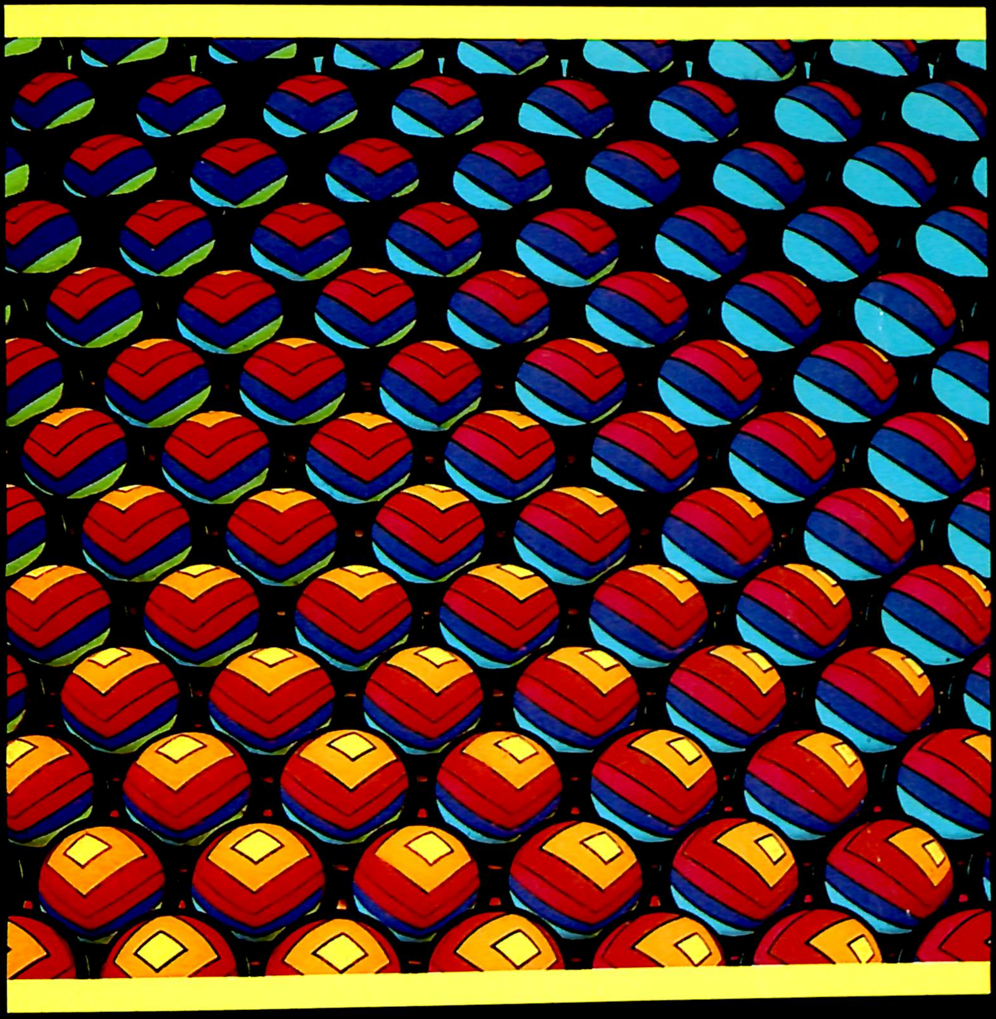
Hi-Res Graphics and Animation Using Assembly Language

The Guide for Apple® II Programmers



includes
double hi-res
for the Apple® IIc
and extended-
memory
Apple® IIe

Leonard I. Malkin, Ph.D.

HAYDEN

[redacted]

Hi-Res Graphics and Animation Using Assembly Language

The Guide for Apple II® Programmers

Hi-Res Graphics and Animation Using Assembly Language

The Guide for Apple II® Programmers

Leonard I. Malkin, Ph.D.

Hayden Book Company
A DIVISION OF HAYDEN PUBLISHING COMPANY, INC.
HASBROUCK HEIGHTS, NEW JERSEY

Acquisitions Editor: KAREN PASTUZYN
Production Editor: ALBERTA BODDY
Design: JIM BERNARD
Cover photo: LOU ODOR
Compositor: McFARLAND GRAPHICS AND DESIGN, INC.
Printer: J.D. LUCAS PRINTING COMPANY

I would like to thank my editor, Karen Pastuzyn, who helped me turn a rough idea into a finished manuscript, and also the reviewers, who lent their assembly language expertise to many parts of the programs.

Library of Congress Cataloging-in-Publication Data

Malkin, Leonard I.
Hi-res graphics and animation using assembly language.

Includes index.
1. Apple II (Computer)—Programming. 2. Computer graphics. 3. Assembler language (Computer program language) I. Title.
QA76.8.A662M35 1985 006.6 85-24752
ISBN 0-8104-6758-5

Printed in the United States of America

2	3	4	5	6	7	8	9	PRINTING
86	87	88	89	90	91	92	93	94	YEAR

To Diane, Sonya, and Joshua,
with whom I can now get reacquainted,
and to my parents, who made me.

EQUIPMENT NEEDED

To use the programs in this book, you will need the following equipment:

- An Apple II Plus, IIe, or IIc (Chapter 12 requires either a IIc or an extended-memory IIe)
- A disk drive
- A monitor (color for Chapter 11 and part of Chapter 12)
- A joystick or paddle
- An assembler (see the *What You Will Need* section)

What You Will Need

If you have an Apple II (II+, IIe, or IIc), and someplace to plug it in, you're practically all set. You will need a disc drive and a display screen, which can be either a black and white or color monitor or television set. Monitors give sharper pictures and are recommended, especially for double hi-res, but television sets are adequate. You should also have a joystick or paddle controls.

You will also need a good assembler. Assemblers are software packages that allow you to write and, more importantly, edit assembly language programs. Strictly speaking, you don't need an assembler to enter the programs in this book (you could use the Apple's resident Monitor or even BASIC), but the level of inconvenience would be unbearably high. Also, for you assembly language beginners out there, don't be lulled by those who may tell you that the Apple's Mini-Assembler or some other simple assembler is sufficient for your needs. The most important characteristic of full feature assemblers is their convenience, not their complexity. To eliminate long hours of needless work, and certainly if you're going to do any serious assembly language programming, a full feature assembler is a necessity. All programs in this book were assembled using the BIG MAC assembler (available from A.P.P.L.E., 290 S.W. 43rd St., Renton, WA 98055; call 1-800-426-3667 to order), but any full feature assembler can be used as they all employ the same basic command set. Among others I can recommend are Orca/M (Hayden Software), Merlin (Southwestern Data Systems), and DOS Tool Kit (Apple Computer, Inc.). These are available in computer stores and are also discounted by mail order firms—check the software ads in any computer magazine. If you don't want to invest in an assembler just now, contact your local Apple user's group—you may be able to borrow an assembler for temporary use.

There are usually some minor differences from assembler to assembler but these are almost always in extra features rather than in the basic system. Features of the BIG MAC assembler used in this book that may not be found in other assemblers are pointed out in the text along with the normal or standard instructions. If you're not going to use BIG MAC, examine your assembler's instruction manual. This, together with an examination of the generated machine code, will tell you what changes, if any, have to be made in the way the assembly code is written.

Introduction

Part One of this book will lead you, step-by-step, through the construction of a single, arcade-type hi-res game written entirely in Apple II assembly language. Each chapter in Part One provides a building block leading to the final game with minimal digressions. Later chapters (Part Two) discuss aspects of hi-res animated graphics important to the subject but not directly related to the game, with suggestions about how to apply these techniques to the game itself or to your own programs.

The game we're going to construct is relatively simple but the program code is not. Hopefully, reading this book will reduce the level of difficulty to manageable proportions. It is written for beginners and experienced users alike and no prior knowledge of assembly language is required. It begins with a discussion of bits and bytes, binary and hexadecimal numbering systems, architecture of the Apple II hi-res screens, use of an assembler, and proceeds with a discussion of drawing and animating shapes, paddle and joystick controls, collision detection, scoring and sound, and finally the game itself. Other topics discussed in both Parts One and Two include animating multiple shapes, drawing over backgrounds, animation in color and in double hi-res color and black and white, advanced paddle and joystick routines, and integrating BASIC with assembly language programs.

Studying this book slowly and methodically will provide you with knowledge of the elements of hi-res game design for the Apple and you will be able to program your own hi-res animation routines in assembly language. However, it should be emphasized that the skills you will acquire have utility far beyond merely designing games. Let me give you a concrete example. I've recently completed an educational program for the Apple II that required moving rather large shapes around the screen and attempts to do this from BASIC using Apple shape tables (we'll discuss these in Chapter 1) were far from satisfactory. The jerky, flickering animation seemed designed to ensure nervous blinking. Using the simple principles described in this book, I was able to produce smooth, professional-looking animation that contributes greatly to the visual appeal of the program, which is one of its strong selling points. So even if game design is not your goal, hi-res animation using assembly language will provide you with an extremely useful tool for a myriad of applications, limited only by your imagination.

Finally, I strongly encourage you to play an active role in the learning process. Do not merely read the text; type in the programs. Try the advanced techniques described in Part Two to modify the game and, above all, develop your own programs. In this way you will learn not only the techniques of hi-res graphics and animation but also many fundamental principles of assembly language programming. Reading about assembly language instructions is one thing but using them in your own programs is another. In the words of an ancient Chinese philosopher,

I hear, and I forget,
I see, and I remember,
I do, and I understand.

Contents

PART ONE

Fundamentals and the Game

1

Why Assembly Language for Hi-Res Animated Graphics?

An English teacher named Bea
Knew the dictionary from A to Z,
But upon buying an Apple
She then had to grapple
With a brand new vocabulary.

Programming in assembly language is not the only way to produce hi-res animated graphics on the Apple II. Applesoft BASIC supports many graphics features that can be quite useful for displaying shapes or moving one or two relatively small shapes around the screen. In fact, it is often convenient to combine graphics from BASIC with assembly language graphic routines, and we will discuss how to do this in Chapter 16. But, as we'll soon see, there are problems associated with using BASIC for graphics programming.

Simple BASIC commands allow one to plot points or lines (and thus shapes) on the hi-res screen and to move them around by erasing and redrawing at a new position. For example, the following BASIC program plots a horizontal line and moves it down one line:

```
10 HGR: REM CLEARS AND DISPLAYS HI-RES SCREEN
20 HCOLOR=3: REM COLOR SET TO WHITE
30 HPLOT 20,20 TO 100,20: REM DRAWS HORIZONTAL LINE
40 HCOLOR=0: HPLOT 20,20 TO 100,20: REM ERASES LINE BY REDRAWING IN BLACK
50 HCOLOR=3: HPLOT 20,21 TO 100,21: REM REDRAWS LINE IN NEW POSITION
```

The line can be made to traverse the screen by continuing the program and changing plot coordinates. One can also draw vertical or diagonal lines and move them across the screen. By specifying different values for HCOLOR, the lines can be drawn in any of the four hi-res colors (blue, orange, violet, and green). This routine is fine for drawing and moving lines, but is far too cumbersome for complicated shapes and entirely inappropriate for rapid and smooth animation—BASIC is just too slow. Consider that even a simple shape may consist of 5 or 10 lines, and moving a shape across the entire hi-res screen involves over 200 draw-erase cycles. Now imagine a routine to move several such shapes at the same time. Attempting to do this in BASIC, in the way described above, would result in an enormous, and enormously difficult to write, program. In addition, the animation would be extremely slow and jerky.

There is yet another method for programming hi-res graphics from BASIC and this involves using Apple shape tables. Details are contained in the Apple BASIC manual so I will touch on the subject only briefly. The instructions for drawing a shape (not the shape itself) are stored somewhere in memory in what are called, appropriately enough, shape tables. A single shape table can contain instructions for more than one shape. For example, to draw the first shape of a shape table, the location of the table is specified by POKEing the appropriate numbers into certain memory locations. Then the color is chosen by assigning a number to HCOLOR, and values for rotation (ROT) and scale (SCALE) are specified. The instruction DRAW 1 AT X,Y will draw the first shape of the table at the coordinates specified by X and Y. By changing the HCOLOR value, the shape can be drawn in different colors. Changing the values for ROT and SCALE allows one to rotate the shape and scale it up in size (although this latter feature is of limited usefulness because the scaling is not proportional). The shape can be erased by the instruction XDRAW 1 AT X,Y or by changing the color to black (HCOLOR = 0) and reDRAWing at X,Y. By erasing and redrawing at different nearby coordinates, the shape can be made to appear to move.

Using shape tables is a neat and convenient way to program hi-res graphics, but there are three problems associated with their use. First, although any of the hi-res colors can be selected, the shape can be only one color—multiple colors in a single shape is not possible. Second, constructing a shape table in the way described in the Apple BASIC manual is a horrendous task. The manual itself recommends using one of the many commercially available utility programs for this purpose—an example is the Apple Mechanic program from Beagle Bros. Such utilities work well (you draw the shape, point by point, and the program assembles it automatically into a shape table) but, as is often the case with someone else's program, you may not be able to get it to do what you want it to do. The Apple Mechanic, for example, limits the overall size of the shape and this may not be appropriate for your needs. Third, smooth and rapid animation with large shapes or with many shapes moving at the same time is not possible using shape tables. The draw, erase, redraw cycles are just too slow, and excessive flickering and jerky movement are the results. Again, as with HPLOTting, shape tables do ~~not~~ have their place (I use them in my own commercial programs), but they do not provide the versatility afforded by assembly language programming.

There are a few graphics utility programs on the market that purport to greatly simplify hi-res animation and they do. But they also, in my hands at least, suffer from many of the problems associated with shape tables and graphics from BASIC and thus, in my opinion, have limited usefulness. Again, using someone else's program almost assuredly will place limits on what you can do. For example, the programs I am familiar with limit the size of the shapes and the number of shapes you can display at any one time. Most have no provision for sound. They are also too slow—the more and larger the shapes, the slower and jerkier the animation. Some of these programs may satisfy your particular needs but don't buy one without return privileges.

The essence of good animation is speed. The illusion of continuous movement can be accomplished only by very rapid draw and erase cycles, especially for large shapes. This also applies in the case of the game we're going to construct, where one desires the illusion of simultaneous movement of multiple shapes. Assembly language provides this speed — in fact, as we'll soon see, assembly language speed is so great that time delays have to be placed in the game program to slow down the action to a reasonable pace.

In addition to speed, assembly language provides the ultimate in versatility. You want to draw and move a shape that takes up half the screen? OK, no problem. How about moving five shapes in different directions at the same time, with sound effects and all possible colors? Also no problem (actually, it is a problem but solvable with assembly language).

Finally, if you're like I am, you want to know and control what's going on. How is your computer drawing and moving all those shapes? Using someone else's program or using BASIC or shape tables tells you very little. Writing your own assembly language programs tells you a great deal.

Speed, versatility, understanding—only assembly language provides this combination of virtues.

2

Bits and Bytes, Sugar and Spice

There once was a fellow named Tex
Whose computer kept him from sex.
When offered a slumber
By a cute little number
He said, "I really prefer binary and hex."

A certain minimal knowledge of binary and hexadecimal numbering systems, the Apple memory map, details of the hi-res screens, and the use of an assembler is necessary before going on to a discussion of assembly language hi-res drawing and animation. Those who know this material can skip to Chapter 3. Those who don't will need to slog their way through this chapter. I'll try to make the slogging as painless as possible.

BINARY NUMBER SYSTEM

Computers operate essentially by using thousands of 2-position switches. Everything a computer does, taking in data (or text, which to a computer is just another form of data), manipulating it, and sending it out to a screen or printer or other device, is all controlled by these switches. A switch can either be on or off (more precisely, high voltage or low voltage). If we assign a 1 and a 0 to these alternate states, we then have a way of representing the status of these switches with numbers. To "talk" to a computer, to tell it what to do, we have to set its switches by talking its language. The only language a computer understands is the language of 0's and 1's, which comprises what is called a binary number system. Higher level computer languages, such as BASIC, use interpretive programs to convert text and decimal number instructions into a binary form. To use lower level languages, such as assembly language, and to understand hi-res graphics, some understanding of the binary system is required.

In any language, all possible words are represented by arranging the alphabet characters in different combinations. Computer "words" are numbers and the computer "alphabet" is 0 and 1. How can just two digits be used to represent more than two numbers? The universally used numbering system is, of course, the decimal system which uses ten digits, 0 to 9, to represent all possible numbers (this is undoubtedly related to the fact that we have ten fingers and toes; if we had only two, we would probably be balancing our checkbooks in binary). We have to realize that the decimal system is just as arbitrary as any other system using any other number of digits. Thus, to understand the binary system requires only an understanding of the principles of the decimal system.

The decimal system works by column assignments. There is no single digit to represent the number ten, so a 1 is placed in a second column, the tens column. Similarly, we represent one hundred by placing a 1 in the third or hundreds column. Each column represents some whole factor of 10.

1000's 10^3	100's 10^2	10's 10^1	1's 10^0	
4	3	2	7	= 4000 + 300 + 20 + 7 = 4327

In the binary system, we can count to one easily enough (zero, one) but there is no single digit to represent the number two so we place a 1 in a second column. Thus, binary 10 = decimal 2 and, it follows, binary 11 = decimal 3. What is decimal 4? Very good. It's binary 100. Thus, the binary system uses columns just like the decimal system except the columns are now factors of two.

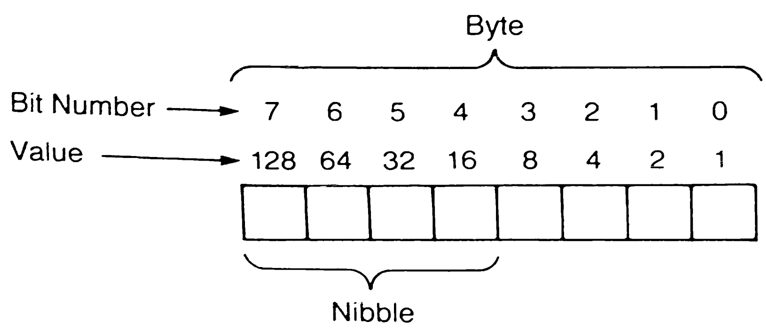
8's 2^3	*4's* 2^1	*2's* 2^1	*1's* 2^0	*Decimal*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Columns can be extended to a 16's column, 32's column, etc. and so given enough columns, we can represent any number by stringing together 0s and 1s.

THE HEXADECIMAL NUMBERING SYSTEM

Writing numbers in binary is obviously a laborious task and is also prone to errors—try copying a string of a hundred 0's and 1's and see how far you get without making a mistake. To avoid these problems, assembly language uses yet another numbering system, the hexadecimal system. An interpreter program converts hexadecimal (or hex for short) numbers into the binary format so the computer can understand what's going on. It will be easier to understand the hexadecimal system if we first discuss some aspects of how the Apple handles numbers.

Each position of a binary number is called a bit. A group of 4 bits is called a nibble and a group of 8 bits is called a byte.



The Apple II is an 8-bit machine; that is, its microprocessor handles 8 bits (1 byte) of data at a time. It's convenient to represent a nibble by a single hex number; thus two hex numbers can represent a single byte. If we look at the table below, we see that a nibble can have values from 0 to 15. We have only ten digits (0-9) to work with, so numbers 10 to 15 are assigned letters A to F (hex numbers are preceded by a $ sign to distinguish them from decimal numbers).

Decimal	*Binary*	*Hex*
0	0 0 0 0	$0
1	0 0 0 1	$1
2	0 0 1 0	$2
3	0 0 1 1	$3
4	0 1 0 0	$4
5	0 1 0 1	$5
6	0 1 1 0	$6
7	0 1 1 1	$7
8	1 0 0 0	$8
9	1 0 0 1	$9
10	1 0 1 0	$A
11	1 0 1 1	$B
12	1 1 0 0	$C
13	1 1 0 1	$D
14	1 1 1 0	$E
15	1 1 1 1	$F

Now we've simplified things somewhat. It's obviously easier to write $F than 1111.

Most of the time we'll be writing numbers as bytes and here the advantage of hex numbers becomes more apparent. To write a byte in hex, we simply assign a hex number to each nibble, e.g.,

Decimal	Binary	Hex
98	$\underbrace{0\ 1\ 1\ 0}_{\$6}\ \underbrace{0\ 0\ 1\ 0}_{\$2}$	$62
198	$\underbrace{1\ 1\ 0\ 0}_{\$C}\ \underbrace{0\ 1\ 1\ 0}_{\$6}$	$C6
255	$\underbrace{1\ 1\ 1\ 1}_{\$F}\ \underbrace{1\ 1\ 1\ 1}_{\$F}$	$FF
1	$\underbrace{0\ 0\ 0\ 0}_{\$0}\ \underbrace{0\ 0\ 0\ 1}_{\$1}$	$01

If you ever feel an irresistible urge to convert hex numbers into binary, you simply take each hex digit and write the corresponding binary nibble. Converting hex to decimal and vice versa is often useful (BASIC uses only decimal numbers). This can be done easily if you understand that the hex system also uses column assignments, just as binary and decimal, but here the columns are factors of 16 (hence the name hexa[6]decimal[10]) because there are 16 digits possible in each column.

16's 16^1	*1's* 16^0	*Hex*	*Decimal*
$1	$0	$10	16
$2	$0	$20	32
$2	$A	$2A	42
$6	$2	$62	98

THE APPLE II MEMORY MAP

The Apple 6502 microprocessor stores numbers in specific locations called memory addresses. Each memory address can hold only one byte. The maximum value of a byte is $FF (11111111 or 255 decimal)—this explains why 255 is the maximum value you can use to POKE to a memory location in BASIC. When these addresses are scanned, a byte is retrieved from each location and depending on the value, a given operation is performed. Memory addresses are accessed by a system that can handle two bytes of data at a time. Two bytes can be represented by four hex numbers, and so a memory address has the general form $NNNN where N equals any hex number. Assemblers always access addresses using the hex format. We can convert memory addresses from hex to decimal (useful when using BASIC and assembly language in the same program) by column assignments; e.g.:

4096's 16^3	*256's* 16^2	*16's* 16^1	*1's* 16^0	*Hex*	*Decimal*
$0	$0	$A	$0	$00A0	160
$0	$8	$0	$0	$0800	2048
$2	$0	$0	$0	$2000	8192
$4	$0	$0	$0	$4000	16384
$6	$0	$0	$0	$6000	24576
$9	$6	$0	$0	$9600	38400
$F	$F	$F	$F	$FFFF	65535

The highest memory address is $FFFF; i.e., all 16 bits are 1. Thus the 6502 microprocessor can access only 65536 addresses ($0000 is the first memory location)—from this comes the term 64K of memory. Apples with 128K of memory switch between two memory banks, each one containing 65536 addresses; Apples with less than 64K of memory have the capability of accessing 65536 addresses—it's just that they're not all there.

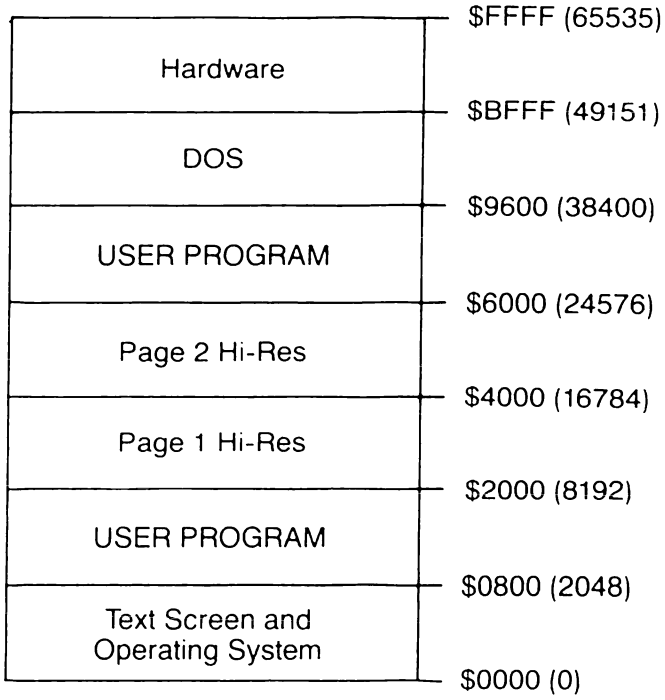
Memory addresses are conveniently divided into what are called pages, each page containing 256 bytes.

Address Bytes	*Hex Address*	*Page Number*
0–255	$0000–$00FF	0
256–511	$0100–$01FF	1
512–767	$0200–$02FF	2
etc.		

Thus, addresses in the range $0000 to $00FF are called zero page addresses. We'll meet up with these later on as they play an important role in some of the assembly language instructions used in our programs.

Memory addresses themselves are often stored at other memory addresses for use in a program. Because an address can store only one byte but is itself represented by two bytes (except for zero page addresses), we have a problem. The solution is to store an address in two locations, one byte in one and one byte in the other. This is done in a particular way. Memory address bytes are divided into two classes, the high order byte (left) and the low order byte (right). For example, $20 is the high byte and $00 the low byte of address $2000. The bytes are stored in consecutive locations, low byte first. We'll learn more about this when we get to our programs in later chapters.

There are several general areas of memory that play a distinctive role in the operation of the Apple II. The following memory map describes and locates some of these functions.

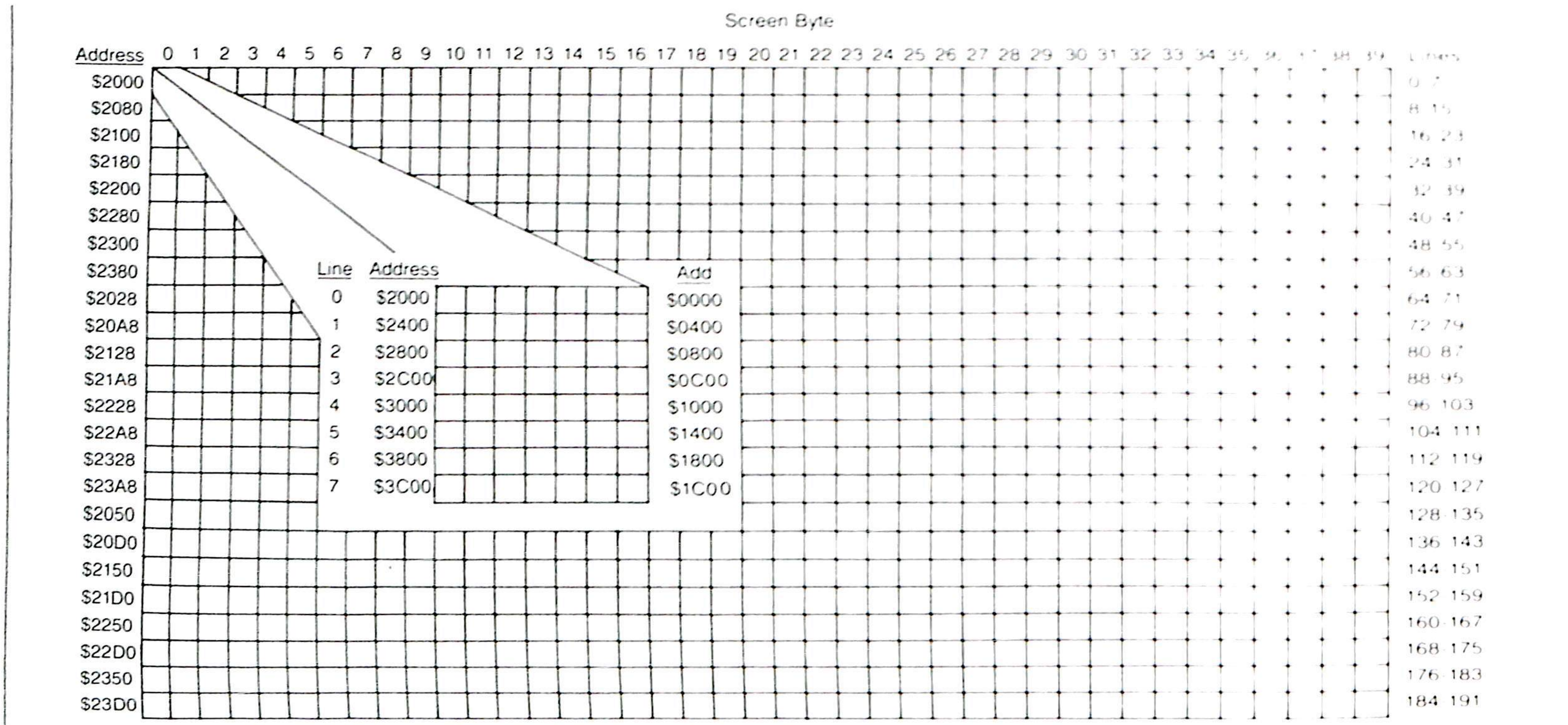


THE HI-RES SCREENS

There are two areas reserved for hi-res graphics, Pages 1 and 2 (these page numbers have nothing to do with the page numbers of memory addresses discussed above). Page 1 occupies an area from $2000 to $3FFF and Page 2 from $4000 to $5FFF. Either page can be used for any hi-res graphics program, the only difference being that Page 1 has the option of displaying full-page graphics or mixed text and graphics, the bottom four lines displaying the text. So if you want to display text and graphics, choose Page 1. For full page graphics, you can choose either page. The only other point to consider in choosing pages is whether you're going to use a BASIC program along with your assembly language program. BASIC requires a continuous stretch of memory, so the page choice determines the maximum length of your BASIC program. For example, if you choose Page 1, you can run BASIC from $0800 to $1FFF or load the BASIC program above Page 1 and run it from $4000 to $9600. This will be discussed in more detail in Chapter 16.

The hi-res screens are divided into screen bytes (horizontal) and lines (vertical). There are 192 lines, numbered 0 to 191, top to bottom, and each line contains 40 screen bytes, numbered 0 to 39 (#$00 to #$27) left to right. Thus there are $40 \times 192 = 7680$ screen byte positions.

In hi-res drawing, only 7 of the 8 bits in a byte are plotted (more on this later) and so each screen byte contains 7 bits, or, as they're called when plotted, pixels (let's get away from computerese and call them dots). Each line then can contain $7 \times 40 = 280$ dots. Therefore a hi-res screen can display up to $280 \times 192 = 53760$ dots; that's why they call it hi-res. So far so good. Everything seems to be in logical order but, of course, there are complications; otherwise, why would you need to read this book? For reasons we won't go into, the Apple designers decided to number hi-res lines in a nonconsecutive fashion. For example, line 0 of the Page 1 screen starts at address $2000 and ends at $2027. You might then expect line 1 to start at $2028, right? Wrong. Line 1 starts at $2400. Line 2 starts at $2800, line 3 at $2C00, and so on, producing quite a scrambled



picture. The same situation holds true for the Page 2 hi-res screen although, of course, with different addresses. There is a method to this mad scramble but we need not concern ourselves with the details because the next chapter will describe a way of accessing any screen position without having to refer to the hi-res screen memory map. The map itself is useful, however, so that you will understand how this is done. In addition, situations may arise where you will want to access particular screen positions directly by referring to the map.

USING AN ASSEMBLER

Finally, we get to the subject of an assembler. As mentioned in the *What You Will Need* section at the beginning of the book, you don't have to use an assembler for your assembly language programs but if you don't, I'll reserve a room for you at the home.

The object of writing an assembly language program is, fittingly enough, to produce object or machine code. Object code is a machine language program that consists entirely of bytes stored at memory addresses. Some of these bytes represent numbers and others represent instructions to the operating system. Object code can look something like this:

```
6000: A9 10
6002: 8D 40 60
```

The code is interpreted as follows. When the program gets to address $6000, byte $A9, an opcode (operation code), tells the computer to store the following number ($10) in the Accumulator, or A, an area for number storage and manipulation in the microprocessor. The first byte ($8D) in the next program line is an opcode that instructs the computer to put the number in the Accumulator at memory address $6040 (note that memory addresses are stored low byte first).

You could enter this code directly from BASIC by POKEing appropriate numbers into appropriate memory locations, remembering first to convert all numbers to decimal. The BASIC program would look like this:

```
POKE 24576, 169
POKE 24577, 16
POKE 24578, 141
POKE 24579, 64
POKE 24580, 96
```

The program could also be entered directly from the Apple's Monitor in this fashion:

```
6000:A9
6001:10
6002:8D
6003:40
6004:60
```

Here is an assembly language code for the same instructions:

```
ORG $6000
LDA #$10
STA $6040
```

ORG $6000 says start the program at address $6000. LDA is a mnemonic for LoaD Accumulator (the Apple 6502 microprocessor uses some 56 mnemonics for assembly language instructions). The # prefix says #$10 is a number, not a memory address. STA is a mnemonic for STore Accumulator and $6040 is the address where #$10 is to be stored. This type of code is called a source code and the assembler, when it is instructed to do so, assembles the source code into the object code and usually will display or print both codes together, one next to the other.

Now, imagine a program hundreds or even thousands of lines long. Obviously, a program written in assembly language is more easily written (and read) than one written in machine language. But assemblers have even more useful features, not the least of which are editing capabilities that allow you to go anywhere in the program and change numbers and lines around without having to reenter the whole thing. In addition, assemblers allow the use of labels and comments, both very useful features.

The source code from most assemblers is divided into several fields or columns. First, a line number is displayed for each instruction. These line numbers are not incorporated into the object code—they are there for editing convenience. The next field is reserved for labels, which are optional. When a region of the program is labeled, it can be accessed by referring to the label rather than to a specific memory location. This not only makes the program more readable but also eliminates the chore of changing instructions to reflect new memory addresses when lines are shifted around. The next field is the command field, which contains the opcode and, if required, the operand, the number or address acted upon by the opcode. Finally, there is the comment field, usually delimited by a semi-colon(;). Comments are similar to REM statements in BASIC and are not incorporated into the object code.

Let's look at a sample program. When the source code is typed in, it will look like this (the field headings are not displayed by the assembler—they are there for your edification):

```
Line   Label     Opcode    Operand       Comments
.....  ........  ........  ..........    ..............

1      *SAMPLE PROGRAM
2                ORG       $6000  ;START PROGRAM AT $6000
3      LOOP      LDA       #$10   ;LOAD A WITH #$10
4                STA       $6040  ;STORE AT $6040
5                JMP       LOOP   ;GO TO LOOP (LINE 3)
```

Line numbers are entered automatically by the assembler. Line 1 demonstrates another feature of assemblers—an entire line can be a comment if delimited by a *. Such lines are not incorporated into the object code. When the command to assemble (usually ASM) is given, the object and source codes are displayed side by side:

```
              1  *SAMPLE PROGRAM
              2           ORG    $6000   ;START PROGRAM AT $6000
6000:A9 10    3  LOOP     LDA    #$10    ;LOAD A WITH #$10
6002:8D 40 60 4           STA    $6040   ;STORE AT $6040
6005:4C 00 60 5           JMP    LOOP    ;GO TO LOOP (LINE 3)
```


The source code and object code are named by you and then saved separately on a disc. The assembler will append a prefix or suffix automatically to one or the other to distinguish which is which. For example, the disc catalog may show the object code as SAMPLE PROGRAM and the source code as SAMPLE PROGRAM.S. This is how programs appear when assembled using the BIG MAC assembler. Other assemblers may do this differently.

The object code is the machine language program we want to run. The source code is not a program and can't be "run" as such. How do we run the program? Object codes are always stored as binary files. To run, we enter BRUN <space> file name (in this case, SAMPLE PROGRAM). This program will be loaded at address $6000 and will run starting from this location. We can also load the program without running it if, for example, we want just to inspect it. The instructions for this are BLOAD <space> file name. To see the program we've loaded, enter the Monitor with CALL-151 and then type 6000L (L for list). The program, along with its assembly language mnemonics but without labels or comments, will be listed starting from $6000. To run the program now we can enter 6000G (G for go to).

Suppose we decide at some later date that $6000 is an inappropriate location for this program because we want to use this area for something else. Let's say we now want to store it at address $4000 instead. We can do this by specifying the address when we BLOAD it, i.e., BLOAD <space> file name,A$4000. The program will now load at $4000 and we can run it from the Monitor by 4000G. What will happen when we run it? Disaster! The reason is that the machine language code is nonrelocatable, that is, it can be run only at the location specified by the ORG statement. To see why this is so, let's look at the code itself. The assembly language instruction in line 5 is JMP LOOP. LOOP is a label that refers to address $6000. Remember that object codes do not deal with labels, only numbers, and so the assembled code for line 5 is 4C 00 60, which is interpreted by the operating system to mean go to address $6000. If the program is loaded at and run from $4000, the 4C 00 60 instruction will be executed faithfully and the program will jump to $6000, which no longer contains the original instruction. Garbage in, garbage out.

It is possible to write relocatable codes, that is, programs that can be loaded anywhere regardless of the address specified by the ORG statement. Sometimes such codes are necessary, but for our purposes this represents just another complication we can do without. If you want to relocate a program, simply call up the source program, change the ORG operand to the new address, and reassemble.

There is one other aspect of assembler use that should be emphasized so I'll mention it here and remind you of it again in later chapters. Assembly language opcodes are entered as 3-letter mnemonics, designed to help you remember what they stand for. Two such opcodes, BCC (Branch on Carry Clear) and BCS (Branch on Carry Set) are often not helpful in this regard. In the BIG MAC and most other full feature assemblers, these opcodes can be replaced by what are called pseudo-opcodes; e.g., BCC can be replaced by BLT (Branch if Less Than) and BCS by BGE (Branch if Greater or Equal). If your assembler doesn't use these pseudo-opcodes, just use BCC and BCS—there is no difference in the assembled program. Purists might argue against the use of pseudo-opcodes because they are not part of the standard Apple instruction set, but they do make programs easier to write and read. I should also mention at this point that the instruction EQU, which is used to assign a label to a memory address, can be replaced in the BIG MAC and other assemblers by the = sign. If your assembler doesn't allow it, use EQU.

If all this is confusing to you, don't worry about it. Get an assembler, read the instructions, look over some of the programs in this book to get a feel for it, and before you know it you'll be a bona fide assembly language programmer. Now, onward and upward (or, in the case of some programs, downward and acrossward).

3

Drawing a Shape on the Hi-Res Screen

Of graphics he certainly could write it;
His talent so great he couldn't hide it.
He plotted a shape
That looked so like a grape
It was all you could do not to byte it.

Drawing a point or a series of points (i.e., a shape) on a hi-res screen involves only three operations:

1. Display the screen.

2. Clear it.

3. Store a byte in a hi-res screen memory location ($2000-$3FFF for Page 1 or $4000-$5FFF for Page 2).

DISPLAYING THE HI-RES SCREEN

In Applesoft BASIC, the command HGR can be used both to clear and to display the Page 1 hi-res screen. Similarly, HGR2 clears and displays hi-res Page 2. We can do this in assembly language by accessing built-in subroutines. For example, JSR $F3E2 is equivalent to HGR and JSR $F3D8 is equivalent to HGR2. This is fine for clearing and displaying a hi-res screen when speed is not required (i.e., at the beginning of a program), but to accomplish this rapidly we need to write our own assembly language routines.

Displaying the hi-res screen of choice involves accessing what are called soft switches. These are certain memory locations that, when accessed, perform the desired function. Accessing a soft switch means either reading from it (PEEKing in BASIC) or writing to it (POKEing in BASIC). It doesn't make any difference which numbers are read from or written to these memory locations. The access process itself is all that's required. Some soft switches require a read, others a write, and some can be accessed either way (details of soft switches can be found in the reference manuals published by Apple for your particular machine). The soft switches of interest for hi-res graphics are the following:

Memory Location of Switch

Decimal	*Hex*	*Function*
49232	$C050	Turns on graphic mode
49239	$C057	Selects hi-res mode
49236	$C054	Selects Page 1
49234	$C052	Selects full page graphics (Page 1)
49237	$C055	Selects Page 2
49235	$C053	Selects mixed text and graphics (Page 1)
49233	$C051	Selects text mode

Arbitrarily, I've decided to use Page 1 with full-screen graphics as the screen of choice for all programs in this book. The switches we want to access then are the first four in the table above. These switches can be accessed by either a read or a write. Try this in BASIC or directly from the keyboard:

```
POKE 49232,0 :POKE 49239,0: POKE 49236,0: POKE 49234,0
```

The Page 1 hi-res screen will be displayed (you will probably see a screen filled with random dots as these instructions, unlike HGR or HGR2, do not clear the hi-res screens). Now, how do we do this in assembly language? The assembly language instruction equivalent to a PEEK in BASIC is LDA, the mnemonic for LoaD Accumulator (the Accumulator is a part of the Apple's 6502 microprocessor that performs most number manipulations). The LDA instruction is used to load the Accumulator with a byte (LDA #$08 loads the number 8 into the Accumulator) or with the contents of a memory location (LDA $2057 loads the Accumulator with the byte stored in location $2057)—note that # preceding a number means it is a number, not a memory location. Because we're simply accessing a soft switch, the particular number loaded into the Accumulator is immaterial.

The assembly language instruction equivalent to a POKE in BASIC is STA (STore Accumulator). This instruction stores the number in the Accumulator in a specified memory location (STA $4097 stores the number in the Accumulator in location $4097). Again, when accessing a soft switch, the particular number is immaterial.

Either LDA or STA can be used to access the soft switches we're interested in but I'm going to use LDA throughout (it appears to be the traditional choice among assembly language programmers). Thus, the assembly language code for displaying the Page 1 hi-res screen with full screen graphics is as follows.

```
]PROGRAM 3-1

:ASM
                      1        ORG  $6000      ;START PROGRAM AT $6000
6000: AD 50 C0        2        LDA  $C050      ;GRAPHICS
6003: AD 57 C0        3        LDA  $C057      ;HI-RES
6006: AD 54 C0        4        LDA  $C054      ;PAGE 1
6009: AD 52 C0        5        LDA  $C052      ;FULL SCREEN GRAPHICS
600C: 60              6        RTS

--End assembly--

13 bytes
```

That's all there is to it! Running this program (see the section in Chapter 2 on using an assembler) will display the Page 1 hi-res screen (again probably with random dots as the screen is not cleared by these instructions). Let's now use a feature of the assembler to make this program more readable. As mentioned previously, we can assign labels to particular memory locations so that the code reads more like text rather than a series of numbers (this is always nice to do so that when you come back to it three months later you won't wonder why in heaven's name you LDAed $C050). Here is the same program with labels for the soft switches (JMP is an instruction equivalent to GOTO in BASIC).

```
]PROGRAM 3-2


:ASM
                   1             ORG   $6000
6000: 4C 03 60     2             JMP   PGM
                   3   GRAPHICS  =     $C050
                   4   HIRES     =     $C057
                   5   PAGE1     =     $C054
                   6   MIXOFF    =     $C052
6003: AD 50 C0     7   PGM       LDA   GRAPHICS
6006: AD 57 C0     8             LDA   HIRES
6009: AD 54 C0     9             LDA   PAGE1
600C: AD 52 C0    10             LDA   MIXOFF
600F: 60          11             RTS

--End assembly--

16 bytes

Symbol table - numerical order:

   PGM      =$6003     GRAPHICS=$C050     MIXOFF   =$C052     PAGE1    =$C054
   HIRES    =$C057
```

Obviously this is a much more readable listing. We're going to use labels as often as we can throughout the book with the idea of eliminating numbers from the source code as much as possible.

CLEARING THE HI-RES SCREEN

Now that we've displayed the hi-res screen, we must clear it before drawing on it. Clearing the screen means turning it all to black, i.e., no dots displayed. The assembly language clear routine is a relatively short program (13 lines), and besides clearing the screen, it also serves as a good example of the use of some common assembly language instructions.

Remember we said before that to draw on a hi-res screen we first display the screen and then store bytes at hi-res screen memory locations. Well, we've already displayed the screen. Now, what bytes do we store and where to clear the screen? It turns out that if you load a hi-res screen location with byte #$00, that portion of the screen will turn to black, i.e., no dots (the relationship of other bytes to what appears on the screen will be dealt with later in this chapter). Thus, to clear the Page 1 hi-res screen we load all the screen locations, from $2000 to $3FFF, with zeros. The following program shows how this is done.

```
]PROGRAM 3-3


:ASM
                 1               ORG   $6000
6000: 4C 03 60   2               JMP   PGM
                 3    GRAPHICS   =     $C050
                 4    HIRES      =     $C057
                 5    PAGE1      =     $C054
                 6    MIXOFF     =     $C052
6003: AD 50 C0   7    PGM        LDA   GRAPHICS
6006: AD 57 C0   8               LDA   HIRES
6009: AD 54 C0   9               LDA   PAGE1
600C: AD 52 C0   10              LDA   MIXOFF
600F: A9 00      11              LDA   #$00          ;CLEAR SCREEN PAGE 1
6011: 85 26      12              STA   $26
6013: A9 20      13              LDA   #$20
6015: 85 27      14              STA   $27
6017: A0 00      15   CLR1       LDY   #$00
6019: A9 00      16              LDA   #$00
601B: 91 26      17   CLR        STA   ($26),Y
601D: C8         18              INY
601E: D0 FB      19              BNE   CLR
6020: E6 27      20              INC   $27
6022: A5 27      21              LDA   $27
6024: C9 40      22              CMP   #$40
6026: 90 EF      23              BLT   CLR1
6028: 60         24              RTS

--End assembly--

41 bytes


Symbol table - numerical order:

   PGM      =$6003     CLR1     =$6017     CLR      =$601B     GRAPHICS=$C050
   MIXOFF   =$C052     PAGE1    =$C054     HIRES    =$C057
```

Let's see how it works (assembly language literates or those simply uninterested can skip to the next section). First, byte #$00 is stored in location $26 (lines 11 and 12). Location $26 is called a zero page address because its actual address is $0026. There's a reason for choosing a zero page address as we'll soon see. Lines 13 and 14 load #$20 into zero page address $27. Line 15 loads #$00 into the Y register (the Apple's microprocessor has two areas other than the Accumulator that can store bytes—the X and Y registers). Line 16 loads the Accumulator with #$00. Line 17 does the real work. It uses a type of command called indirect indexing, which works only with the Y register and a zero page address (hence choosing a zero page address to begin with). STA ($26),Y says take the contents of the Accumulator (#$00 from line 16) and store it in a memory address calculated as follows: go to location $26 to get the low byte of the address and then get the high byte from the next zero page location, i.e., $27, add the contents of the Y register to get the final address.

STA ($26),Y → LOCATION / CONTENTS

LOCATION	CONTENTS
$26	#$00
$27	#$20

ACCUMULATOR #$00

$2000 + Y = $2000

Y = #$00

Note what has happened. A zero has been stored at location $2000, the first location of hi-res screen Page 1, turning it black. We're on our way! Line 18 (INY) now increments the contents of the Y register by one. Y now contains #$01. Line 19 says if Y has not yet reached zero (incrementing the maximum value [#$FF] by one results in #$00), branch back to CLR (line 17). Line 17 now calculates the new address as $2001.

STA ($26),Y → LOCATION / CONTENTS

LOCATION	CONTENTS
$26	#$00
$27	#$20

ACCUMULATOR #$00

$2000 + Y = $2001

Y = #$01

Now we've blacked out the next screen location at $2001. This continues until Y is incremented to zero, thus blacking out 256 bytes. Then the number located in address $27 is incremented by one (line 20). Next we do a comparison to see if we're finished. We load the Accumulator with the byte in $27 and compare it to #$40 (line 22). If the Accumulator contains #$40 we want to stop because this will get us into the Page 2 hi-res screen. The command in line 23 (BLT, Branch if Less Than, a code that can be used by some assemblers in place of the standard BCC, Branch on Carry Clear) says branch or jump to CLR1 if the Accumulator byte is less than #$40. If it is #$40, the branch is not taken and the program ends. When we branch to CLR1, we load Y again with #$00 and line 17 puts a zero at location $2100.

STA ($26),Y → LOCATION / CONTENTS

LOCATION	CONTENTS
$26	#$00
$27	#$21

ACCUMULATOR #$00

$2100 + Y = $2100

Y = #$00

Each time 256 bytes are blacked out, $27 is incremented by one and a new page of memory is selected.

```
Determined by $27 ---> | $20 | 00 ----- 256 bytes (determined by Y) ----------- to $20FF
                       | $21 | 00 ------------------------------------------------ to $21FF
                       | $22 | 00 ------------------------------------------------ to $22FF
                         :     :
                       | $3F | 00 ------------------------------------------------ to $3FFF
                       | $40 | 00 ----- Stop — beginning of page 2 hi-res -----
```

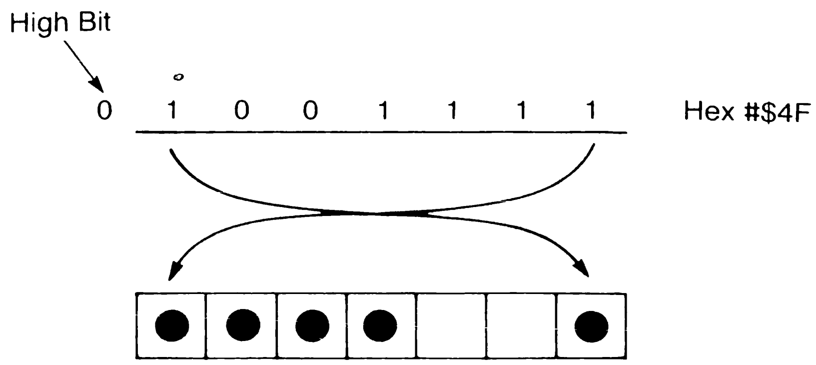
This whole routine takes less than a tenth of a second. Talk about assembly language speed! By the way, if you want to clear hi-res Page 2, place #$40 in line 13 and #$60 in line 22. The screen addresses will then be $4000 to $5FFF.

DRAWING A SHAPE

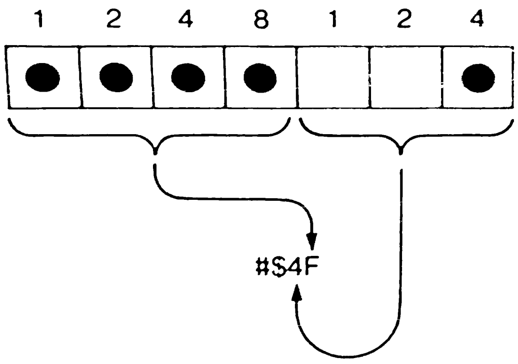
Now that we've displayed the screen and cleared it, let's draw something on it (about time, eh?).

We've seen that if we store a zero at a hi-res screen location, that location turns black. The heart of hi-res drawing is the fact that if we write any byte other than zero to the screen, dots will appear (actually, storing byte #$80 will also produce no dots—this is a complication we don't need, right? We'll discuss why this happens below). Let's now discuss the relationship of bytes to dot patterns. The details are a bit messy but the application is easy.

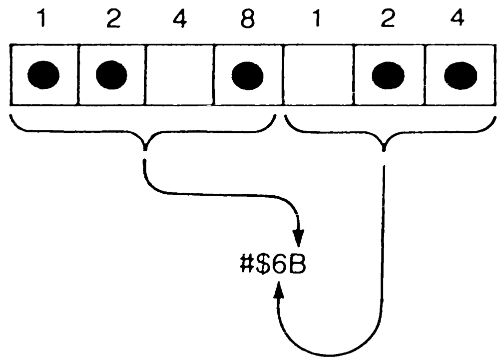
Remember that a byte is a series of 8 bits, each one of which can be off (0) or on (1). You guessed it! If a bit is 0, the screen is black at that point; if a bit is 1, a dot is turned on. But there are complications (you wouldn't want this to be too easy, would you, else how could you impress your friends?). First, only 7 of the 8 bits determine a dot pattern. The leftmost or most significant bit (also called the high bit) is used to select colors for the byte (more on this in a later chapter). This is why storing byte #$80 will produce no dots. For now, we'll always use 0 as the high bit. Second, the remaining 7 bits are plotted backwards! Why? Don't ask. Let's just see how it works. Here is a byte and the dot pattern that results when this byte is sent to a hi-res screen location.



How does one convert a dot pattern to a byte? Don't fret. It's easy. Place the desired dot pattern in 7 boxes and number the boxes as shown.



Take the rightmost 3 bits and convert them to a hex number. This is the first number of the byte. Then do the same for the leftmost 4 bits. This gives you the second number of the byte. You now have the byte, #$4F, that will give you the desired dot pattern. Let's try another example.



Got it? OK. Let's now write a program that will display the dot pattern in the last example, i.e., the one specified by #$6B. We're going to put this byte in the first line (line 0) and the first byte (byte 0) of the hi-res screen Page 1 (location $2000), which is in the upper left-hand corner of the screen. (Be careful to distinguish between the screen byte, which is the location of the horizontal column [0-39 across the screen] where the shape is to be drawn, and the shape byte, which is the byte that determines the dot pattern.)

```
]PROGRAM 3-4


:ASM

                                 ORG  $6000
6000: 4C 03 60                   JMP  PGM
                 GRAPHICS =           $C050
                 HIRES    =           $C057
                 PAGE1    =           $C054
                 MIXOFF   =           $C052
6003: AD 50 C0   PGM         LDA  GRAPHICS
6006: AD 57 C0               LDA  HIRES
6009: AD 54 C0               LDA  PAGE1
600C: AD 52 C0               LDA  MIXOFF
600F: A9 00                  LDA  #$00          ;CLEAR SCREEN PAGE 1
6011: 85 26                  STA  $26
6013: A9 20                  LDA  #$20
6015: 85 27                  STA  $27
```

```
6017: A0 00      CLR1    LDY  #$00
6019: A9 00              LDA  #$00
601B: 91 26      CLR     STA  ($26),Y
601D: C8                 INY
601E: D0 FB              BNE  CLR
6020: E6 27              INC  $27
6022: A5 27              LDA  $27
6024: C9 40              CMP  #$40
6026: 90 EF              BLT  CLR1
6028: A9 6B              LDA  #$6B
602A: 8D 00 20           STA  $2000       ;PLOT BYTE
602D: 60                 RTS

--End assembly--

46 bytes

Symbol table - numerical order:

   PGM     =$6003     CLR1    =$6017     CLR     =$601B     GRAPHICS=$C050
   MIXOFF  =$C052     PAGE1   =$C054     HIRES   =$C057
```

We've now drawn our first shape; admittedly, it's not much of a shape but we have to start somewhere (actually it does look something like a far-away bird or maybe an airplane—it helps to have imagination in this business). Let's get more ambitious now and draw something more interesting, say, a person. The shape will be 1-byte wide by 6-lines deep. Here is the dot pattern, the corresponding bytes, and the line addresses where the bytes will be drawn.

1	2	4	8	1	2	4	Shape Byte	Line Address
			●				#$08	$2000
	●	●	●	●	●		#$3E	$2400
●		●	●	●		●	#$5D	$2800
		●	●	●			#$1C	$2C00
		●		●			#$14	$3000
	●				●		#$22	$3400

```
]PROGRAM 3-5
:ASM
                                ORG  $6000
6000: 4C 03 60                  JMP  PGM
                    GRAPHICS    =    $C050
                    HIRES       =    $C057
                    PAGE1       =    $C054
                    MIXOFF      =    $C052
6003: AD 50 C0      PGM         LDA  GRAPHICS
6006: AD 57 C0                  LDA  HIRES
6009: AD 54 C0                  LDA  PAGE1
600C: AD 52 C0                  LDA  MIXOFF
```

```
600F: A9 00            LDA  #$00        ;CLEAR SCREEN PAGE 1
6011: 85 26            STA  $26
6013: A9 20            LDA  #$20
6015: 85 27            STA  $27
6017: A0 00     CLR1   LDY  #$00
6019: A9 00            LDA  #$00
601B: 91 26     CLR    STA  ($26),Y
601D: C8               INY
601E: D0 FB            BNE  CLR
6020: E6 27            INC  $27
6022: A5 27            LDA  $27
6024: C9 40            CMP  #$40
6026: 90 EF            BLT  CLR1
6028: A9 08            LDA  #$08        ;DRAW SHAPE
602A: 8D 00 20         STA  $2000
602D: A9 3E            LDA  #$3E
602F: 8D 00 24         STA  $2400
6032: A9 5D            LDA  #$5D
6034: 8D 00 28         STA  $2800
6037: A9 1C            LDA  #$1C
6039: 8D 00 2C         STA  $2C00
603C: A9 14            LDA  #$14
603E: 8D 00 30         STA  $3000
6041: A9 22            LDA  #$22
6043: 8D 00 34         STA  $3400
6046: 60               RTS

--End assembly--

71 bytes


Symbol table - numerical order:

   PGM     =$6003    CLR1    =$6017    CLR     =$601B    GRAPHICS=$C050
   MIXOFF  =$C052    PAGE1   =$C054    HIRES   =$C057
```

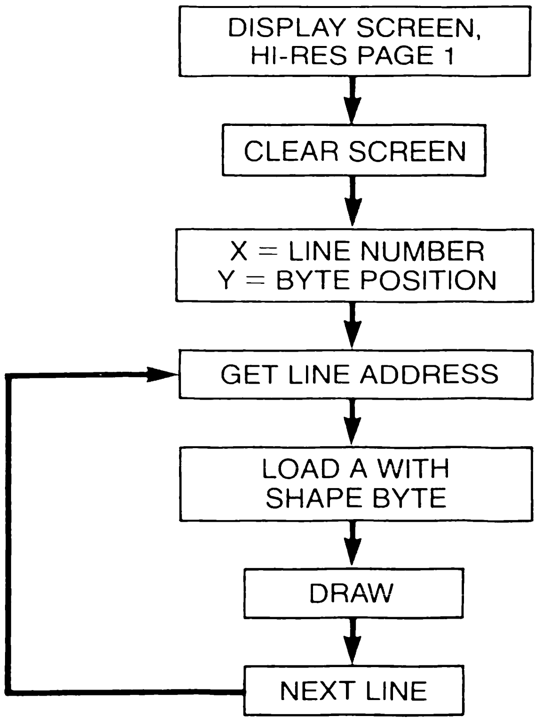

We can put the shape anywhere on the hi-res screen by changing the screen locations. For example, if we want to plot it one byte over (one byte from the left screen border), the addresses would be $2001, $2401, $2801, etc.

LINE ADDRESS TABLES

There's nothing wrong with this program (it works) but it doesn't address (pardon the pun) the major headache in hi-res plotting, i.e., calculating line addresses. We would like to plot the shape from, say, lines 0 to 5 without bothering about the nonconsecutive nature of the screen line addresses. With the procedure I'm about to describe, one can plot a shape at any line and byte position without having to refer to a huge map of all 7680 screen positions. This will become especially important when we deal with animation, which involves moving shapes around the screen. There is more than one way to solve this problem, but the easiest and fastest way is to use table look-ups. The high byte and low byte of each line address is stored in tables. A line number from 0 to 191 is specified; by looking up the table, the correct line address is retrieved.

The byte position (0-39) also is specified and added to the line address to get the correct screen position. Let's see how it works (see Program 3-6).

There are two tables, one labeled HI for the high bytes and one labeled LO for the low bytes. Each table is 192 bytes long for the 192 line addresses. (The BIG MAC Assembler and some others allow the entry of hex numbers without prescripts using the HEX command; some assemblers do not support this instruction and require the code DFB #$20, #$24, #$28, etc. The ORCA/M assembler uses a DC H' directive; refer to your assembler's instructions.) Suppose we want to plot our man shape at byte 0, lines 0 to 5 as before. We'll use the Y register to hold the byte position and the X register to hold the line position.




```
]PROGRAM 3-6
:ASM
                            ORG   $6000
6000: 4C 03 60              JMP   PGM
                GRAPHICS  =     $C050
                HIRES     =     $C057
                PAGE1     =     $C054
                MIXOFF    =     $C052
6003: AD 50 C0  PGM       LDA   GRAPHICS
6006: AD 57 C0            LDA   HIRES
6009: AD 54 C0            LDA   PAGE1
600C: AD 52 C0            LDA   MIXOFF
600F: A9 00               LDA   #$00        ;CLEAR SCREEN PAGE 1
6011: 85 26               STA   $26
6013: A9 20               LDA   #$20
6015: 85 27               STA   $27
6017: A0 00     CLR1      LDY   #$00
6019: A9 00               LDA   #$00
601B: 91 26     CLR       STA   ($26),Y
601D: C8                  INY
601E: D0 FB               BNE   CLR
6020: E6 27               INC   $27
6022: A5 27               LDA   $27
```

```
6024: C9 40              CMP  #$40
6026: 90 EF              BLT  CLR1
           ******************
6028: A2 00              LDX  #$00         ;LINE NUMBER
602A: A0 00              LDY  #$00         ;BYTE NUMBER
602C: BD 86 60           LDA  HI,X         ;GET LINE ADDRESS
602F: 85 77              STA  $77
6031: BD 46 61           LDA  LO,X
6034: 85 76              STA  $76
6036: A9 08              LDA  #$08
6038: 91 76              STA  ($76),Y      ;PLOT
603A: E8                 INX               ;NEXT LINE
603B: BD 86 60           LDA  HI,X
603E: 85 77              STA  $77
6040: BD 46 61           LDA  LO,X
6043: 85 76              STA  $76
6045: A9 3E              LDA  #$3E
6047: 91 76              STA  ($76),Y      ;PLOT
6049: E8                 INX               ;NEXT LINE
604A: BD 86 60           LDA  HI,X
604D: 85 77              STA  $77
604F: BD 46 61           LDA  LO,X
6052: 85 76              STA  $76
6054: A9 5D              LDA  #$5D
6056: 91 76              STA  ($76),Y      ;PLOT
6058: E8                 INX               ;NEXT LINE
6059: BD 86 60           LDA  HI,X
605C: 85 77              STA  $77
605E: BD 46 61           LDA  LO,X
6061: 85 76              STA  $76
6063: A9 1C              LDA  #$1C
6065: 91 76              STA  ($76),Y      ;PLOT
6067: E8                 INX               ;NEXT LINE
6068: BD 86 60           LDA  HI,X
606B: 85 77              STA  $77
606D: BD 46 61           LDA  LO,X
6070: 85 76              STA  $76
6072: A9 14              LDA  #$14
6074: 91 76              STA  ($76),Y      ;PLOT
6076: E8                 INX               ;NEXT LINE
6077: BD 86 60           LDA  HI,X
607A: 85 77              STA  $77
607C: BD 46 61           LDA  LO,X
607F: 85 76              STA  $76
6081: A9 22              LDA  #$22
6083: 91 76              STA  ($76),Y      ;PLOT
6085: 60                 RTS
6086: 20 24 28       HI  HEX  2024282C3034383C ; HIGH BYTE LINE ADDRESSES
6089: 2C 30 34 38 3C
608E: 20 24 28           HEX  2024282C3034383C
6091: 2C 30 34 38 3C
6096: 21 25 29           HEX  2125292D3135393D
6099: 2D 31 35 39 3D
609E: 21 25 29           HEX  2125292D3135393D
60A1: 2D 31 35 39 3D
60A6: 22 26 2A           HEX  22262A2E32363A3E
60A9: 2E 32 36 3A 3E
60AE: 22 26 2A           HEX  22262A2E32363A3E
60B1: 2E 32 36 3A 3E
60B6: 23 27 2B           HEX  23272B2F33373B3F
60B9: 2F 33 37 3B 3F
```

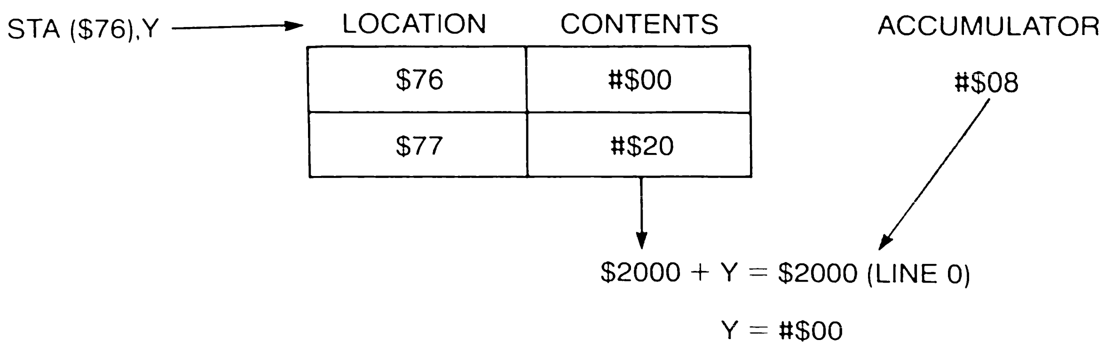
```
60BE: 23 27 2B  76          HEX  23272B2F33373B3F
60C1: 2F 33 37 3B 3F
60C6: 20 24 28  77          HEX  2024282C3034383C
60C9: 2C 30 34 38 3C
60CE: 20 24 28  78          HEX  2024282C3034383C
60D1: 2C 30 34 38 3C
60D6: 21 25 29  79          HEX  2125292D3135393D
60D9: 2D 31 35 39 3D
60DE: 21 25 29  80          HEX  2125292D3135393D
60E1: 2D 31 35 39 3D
60E6: 22 26 2A  81          HEX  22262A2E32363A3E
60E9: 2E 32 36 3A 3E
60EE: 22 26 2A  82          HEX  22262A2E32363A3E
60F1: 2E 32 36 3A 3E
60F6: 23 27 2B  83          HEX  23272B2F33373B3F
60F9: 2F 33 37 3B 3F
60FE: 23 27 2B  84          HEX  23272B2F33373B3F
6101: 2F 33 37 3B 3F
6106: 20 24 28  85          HEX  2024282C3034383C
6109: 2C 30 34 38 3C
610E: 20 24 28  86          HEX  2024282C3034383C
6111: 2C 30 34 38 3C
6116: 21 25 29  87          HEX  2125292D3135393D
6119: 2D 31 35 39 3D
611E: 21 25 29  88          HEX  2125292D3135393D
6121: 2D 31 35 39 3D
6126: 22 26 2A  89          HEX  22262A2E32363A3E
6129: 2E 32 36 3A 3E
612E: 22 26 2A  90          HEX  22262A2E32363A3E
6131: 2E 32 36 3A 3E
6136: 23 27 2B  91          HEX  23272B2F33373B3F
6139: 2F 33 37 3B 3F
613E: 23 27 2B  92          HEX  23272B2F33373B3F
6141: 2F 33 37 3B 3F
6146: 00 00 00  93   LO     HEX  0000000000000000 ; LOW BYTE LINE ADDRESSES
6149: 00 00 00 00 00
614E: 80 80 80  94          HEX  8080808080808080
6151: 80 80 80 80 80
6156: 00 00 00  95          HEX  0000000000000000
6159: 00 00 00 00 00
615E: 80 80 80  96          HEX  8080808080808080
6161: 80 80 80 80 80
6166: 00 00 00  97          HEX  0000000000000000
6169: 00 00 00 00 00
616E: 80 80 80  98          HEX  8080808080808080
6171: 80 80 80 80 80
6176: 00 00 00  99          HEX  0000000000000000
6179: 00 00 00 00 00
617E: 80 80 80  100         HEX  8080808080808080
6181: 80 80 80 80 80
6186: 28 28 28  101         HEX  2828282828282828
6189: 28 28 28 28 28
618E: A8 A8 A8  102         HEX  A8A8A8A8A8A8A8A8
6191: A8 A8 A8 A8 A8
6196: 28 28 28  103         HEX  2828282828282828
6199: 28 28 28 28 28
619E: A8 A8 A8  104         HEX  A8A8A8A8A8A8A8A8
61A1: A8 A8 A8 A8 A8
61A6: 28 28 28  105         HEX  2828282828282828
61A9: 28 28 28 28 28
61AE: A8 A8 A8  106         HEX  A8A8A8A8A8A8A8A8
```

```
61B1: A8 A8 A8 A8 A8
61B6: 28 28 28   107       HEX  2828282828282828
61B9: 28 28 28 28 28
61BE: A8 A8 A8   108       HEX  A8A8A8A8A8A8A8A8
61C1: A8 A8 A8 A8 A8
61C6: 50 50 50   109       HEX  5050505050505050
61C9: 50 50 50 50 50
61CE: D0 D0 D0   110       HEX  D0D0D0D0D0D0D0D0
61D1: D0 D0 D0 D0 D0
61D6: 50 50 50   111       HEX  5050505050505050
61D9: 50 50 50 50 50
61DE: D0 D0 D0   112       HEX  D0D0D0D0D0D0D0D0
61E1: D0 D0 D0 D0 D0
61E6: 50 50 50   113       HEX  5050505050505050
61E9: 50 50 50 50 50
61EE: D0 D0 D0   114       HEX  D0D0D0D0D0D0D0D0
61F1: D0 D0 D0 D0 D0
61F6: 50 50 50   115       HEX  5050505050505050
61F9: 50 50 50 50 50
61FE: D0 D0 D0   116       HEX  D0D0D0D0D0D0D0D0
6201: D0 D0 D0 D0 D0

--End assembly--

518 bytes

Symbol table - numerical order:

  PGM     =$6003   CLR1       =$6017   CLR     =$601B   HI      =$6086
  LO      =$6146   GRAPHICS=$C050      MIXOFF  =$C052   PAGE1   =$C054
  HIRES   =$C057
```


Let's look at the program starting from line 25.

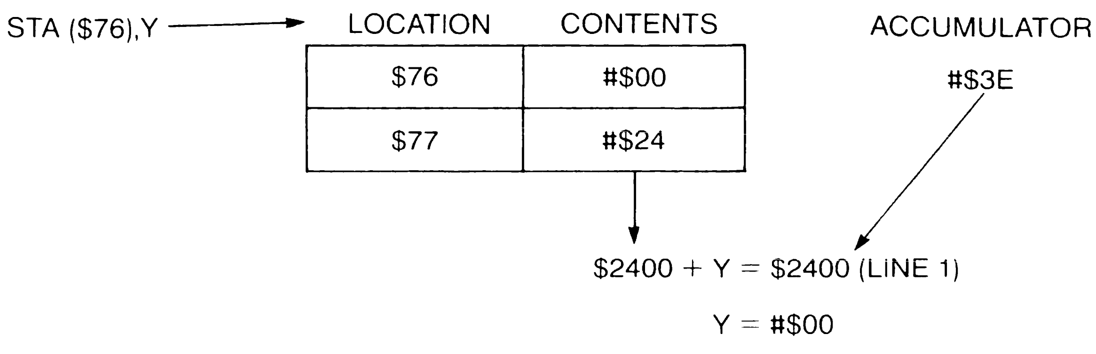
LDX #$00	Line number in X register
LDY #$00	Byte number in Y register
LDA HI,X	This instruction is called absolute indexing. The Accumulator is loaded with the byte found in location HI + X (remember that HI is a label for a particular address). Because X = 0, the first byte in the HI table (#$20) is loaded into the Accumulator.
STA $77	The contents of the Accumulator (#$20) are placed in a zero page location.
LDA LO,X	The Accumulator is loaded with the low byte of the line address, i.e., the byte in LO + X. Because X = 0, the first byte in the LO table (#$00) is loaded into the Accumulator.
STA $76	#$00 is placed in another zero page location.
	$76 and $77 now contain the low and high bytes of the address of line 0 ($2000).
LDA #$08	The first shape byte to be plotted is put into the Accumulator.

STA ($76),Y We've seen this instruction before in the clear screen routine. It stores the Accumulator contents at a screen address retrieved from the contents of zero page addresses $76 and $77 plus Y, the byte position.



We've now plotted the first shape byte at line 0, byte 0. The second shape byte now goes on line 1. To plot on this line, we increment X by one and repeat the above steps with the next shape byte.

INX	X now contains #$01
LDA HI,X	Loads the Accumulator with the second byte in table HI (HI + 1 = #$24)
STA $77	$77 now contains #$24.
LDA LO,X	Loads the Accumulator with the second byte in table LO (LO + 1 = #$00).
STA $76	$76 now contains #$00.
LDA #$3E	Loads the Accumulator with the second shape byte.
STA ($76),Y	The second shape byte is plotted at $2400 + Y = $2400 (line 1).



These steps are repeated until all the shape bytes are drawn. We can change the byte and line locations by putting different values in the Y and X registers. For example, to plot the shape starting at screen line 5 and screen byte 4, place 5 in X and 4 in Y. LDA HI,X and LDA LO,X retrieves the line address $3400. STA ($76),Y adds 4 to this address to get the desired screen position, $3404.

```
Symbol table - numerical order:

    LOW      =$1A        HIGH      =$1B       BYTE      =$6003     LINE      =$6004
    DEPTH    =$6005      XCOUNT    =$6006     PGM       =$6007     CLR1      =$601B
    CLR      =$601F      DRAW      =$6041     SHAPE     =$6068     HI        =$606E
    LO       =$612E      GRAPHICS=$C050       MIXOFF    =$C052     PAGE1     =$C054
    HIRES    =$C057
```

HI and **LO** refer to the tables in Program 3-6.

Let's examine the program in some detail, because some new elements of assembler use have been added. We need to reserve some space in the program to hold the values for byte, line, depth of shape, and XCOUNT (the use of XCOUNT will be described below). This is done by using the code DS for Defined Storage. Thus, BYTE DS 1 will reserve one memory location somewhere between $6000 and PGM and label it BYTE (the precise location is displayed in the Symbol Table at the end of the program listing). Also, in keeping with our desire to remove numbers from the source code, we define zero page addresses $1A as LOW and $1B as HIGH, and use these labels also in the clear screen routine. (Using $1A and $1B as zero page addresses ensures no conflict with DOS commands or any BASIC program we might want to integrate with our assembly language program—see Chapter 16).

We first enter the initial values for line, byte, and depth of shape (lines 31 to 37). Note that the shape depth is added to the starting line number (lines 33 to 35) so that DEPTH will contain the value #$05 + #$06 = #$0B (ADC means ADd with Carry and must always be preceded with CLC, CLear Carry). In the DRAW routine, Y is loaded with the screen byte (line 40) and X with the starting line (line 41). XCOUNT is initially set to zero (lines 38 to 39). Lines 42 to 45 get the line address for the first line to be plotted.

We now need another counter to access the bytes in the shape table but there are no more available—A, X, and Y are being used already. To get around this, we load X temporarily with the value in XCOUNT and use XCOUNT as the shape table counter (X is reloaded with the line number by line 41). Thus, LDA SHAPE,X (line 47) loads A, the Accumulator, with the first byte of the shape table, because X = 0 from the initial value of XCOUNT. STA (LOW),Y (line 48) then plots the first shape byte at line 5, byte 4. XCOUNT is incremented by one (line 49) and now contains the value #$01. LINE is also incremented by one (line 50); it now contains the value #$06. This new line number is now compared to the value in DEPTH (line 52).

To do a CMP comparison, you must first load A with the number to be compared to (line 51). BLT DRAW (line 53) is an instruction that says if the number in A (the line number) is less than the number in DEPTH, go back to DRAW and continue drawing. At DRAW, X is loaded with the new line number (#$06) and a new address is obtained from the HI and LO tables. X is loaded with the new XCOUNT (#$01) and LDA SHAPE,X loads A with the second byte of the shape table. This is then plotted at the new line by STA (LOW),Y. Thus, the second shape byte is plotted at screen line 6 and screen byte 4 (in this program, the screen byte isn't changed). This whole process is then repeated until the last line plotted is equal to DEPTH. Then the branch at line 53 is not taken and the program ends.

Compared to the previous program, this program is not only shorter but also easier to read and manipulate. For example, if we don't like the way the shape looks, we can simply change numbers in the shape table. For larger programs with multiple shapes, the advantage of using shape tables becomes even more apparent.

DRAWING SHAPES WIDER THAN ONE BYTE

We've one more topic to discuss before we leave this chapter. Up to now, we've only plotted shapes of width one screen byte or less. Suppose we want to plot a shape that extends over two bytes or more. A slight change in the drawing routine is required. The following program (Program 3-8) plots the shape of a plane that is 2-bytes wide and 5-lines deep.

1	2	4	8	1	2	4	1	2	4	8	1	2	4	SHAPE BYTES Screen Byte 1	Screen Byte 2
●	●													#$03	#$00
●	●													#$03	#$00
●		●	●	●	●	●	●	●	●	●	●	●		#$7D	#$3F
●													●	#$01	#$40
●	●	●	●	●	●	●	●	●	●	●	●	●		#$7F	#$3F

The order of drawing will be:

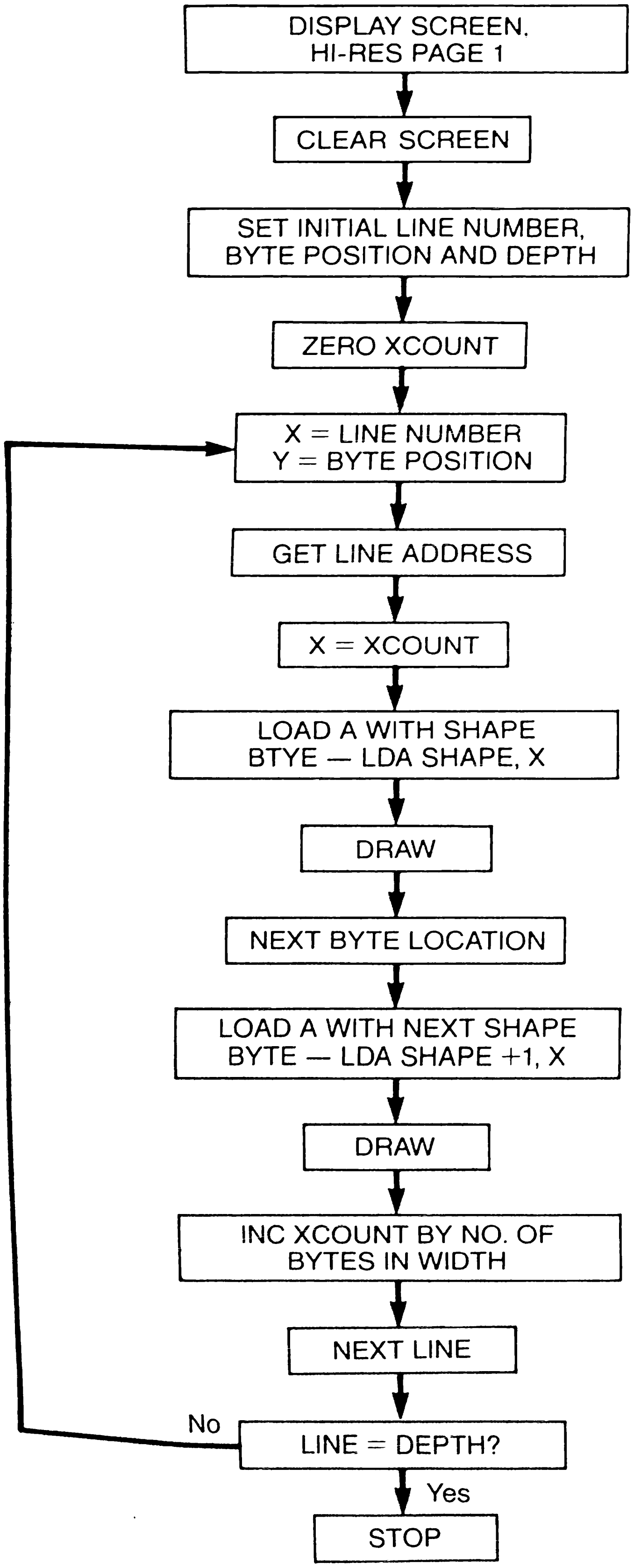
line 1, first screen byte, second screen byte

line 2, first screen byte, second screen byte

line 3, first screen byte, second screen byte, etc.

Thus, the order of shape bytes in the shape table is 03 00 03 00 7D 3F 01 40 7F 3F.

In the DRAW routine, we get the address of the first screen line and first screen byte and plot the first byte of the shape table. Then, on the same line, we increment Y (line 49) so that the next plot (STA (LOW),Y) will be at the second screen byte. LDA SHAPE+1,X (line 50) retrieves the second byte of the shape table for this plot. XCOUNT is then incremented by the number of bytes in the shape width; in this case, two. We then go to the next line by incrementing LINE (line 54) and, because the shape isn't finished yet, we go back to DRAW to reset the screen byte to its initial value (line 40) and obtain the new line address. Now LDA SHAPE,X will get the third shape byte because X = 2 from XCOUNT. INY gets us to the next screen byte and LDA SHAPE+1,X retrieves the fourth shape byte. This continues until CMP DEPTH tells us the shape is finished.


DISPLAY SCREEN,
HI-RES PAGE 1
CLEAR SCREEN
SET INITIAL LINE NUMBER,
BYTE POSITION AND DEPTH
ZERO XCOUNT
X = LINE NUMBER
Y = BYTE POSITION
GET LINE ADDRESS
X = XCOUNT
LOAD A WITH SHAPE
BTYE — LDA SHAPE, X
DRAW
NEXT BYTE LOCATION
LOAD A WITH NEXT SHAPE
BYTE — LDA SHAPE +1, X
DRAW
INC XCOUNT BY NO. OF
BYTES IN WIDTH
NEXT LINE
No
LINE = DEPTH?
Yes
STOP

```
]PROGRAM 3-8
:ASM
                          ORG  $6000
6000: 4C 07 60            JMP  PGM
                 BYTE     DS   1
                 LINE     DS   1
                 DEPTH    DS   1
                 XCOUNT   DS   1
                 GRAPHICS =    $C050
                 HIRES    =    $C057
                 PAGE1    =    $C054
                 MIXOFF   =    $C052
                 HIGH     =    $1B
                 LOW      =    $1A
6007: AD 50 C0   PGM      LDA  GRAPHICS
600A: AD 57 C0            LDA  HIRES
600D: AD 54 C0            LDA  PAGE1
6010: AD 52 C0            LDA  MIXOFF
6013: A9 00               LDA  #$00       ;CLEAR SCREEN PAGE 1
6015: 85 1A               STA  LOW
6017: A9 20               LDA  #$20
6019: 85 1B               STA  HIGH
601B: A0 00      CLR1     LDY  #$00
601D: A9 00               LDA  #$00
601F: 91 1A      CLR      STA  (LOW),Y
6021: C8                  INY
6022: D0 FB               BNE  CLR
6024: E6 1B               INC  HIGH
6026: A5 1B               LDA  HIGH
6028: C9 40               CMP  #$40
602A: 90 EF               BLT  CLR1
                 *****************
602C: A9 05               LDA  #$05
602E: 8D 04 60            STA  LINE       ;LINE NUMBER
6031: 18                  CLC
6032: 69 05               ADC  #$05
6034: 8D 05 60            STA  DEPTH      ;ADD DEPTH OF SHAPE
6037: A9 04               LDA  #$04
6039: 8D 03 60            STA  BYTE       ;BYTE
603C: A9 00               LDA  #$00
603E: 8D 06 60            STA  XCOUNT     ;ZERO XCOUNT
6041: AC 03 60   DRAW     LDY  BYTE       ;BYTE IN Y REGISTER
6044: AE 04 60            LDX  LINE       ;LINE IN X REGISTER
6047: BD 7B 60            LDA  HI,X       ;GET LINE ADDRESS
604A: 85 1B               STA  HIGH
604C: BD 3B 61            LDA  LO,X
604F: 85 1A               STA  LOW
6051: AE 06 60            LDX  XCOUNT     ;LOAD X WITH XCOUNT
6054: BD 71 60            LDA  SHAPE,X    ;GET SHAPE BYTE
6057: 91 1A               STA  (LOW),Y    ;PLOT
6059: C8                  INY             ;NEXT BYTE
605A: BD 72 60            LDA  SHAPE+1,X  ;NEXT SHAPE BYTE
605D: 91 1A               STA  (LOW),Y    ;PLOT
605F: EE 06 60            INC  XCOUNT     ;INC XCOUNT BY NO. OF
6062: EE 06 60            INC  XCOUNT        BYTES IN SHAPE WIDTH
6065: EE 04 60            INC  LINE       ;NEXT LINE
6068: AD 04 60            LDA  LINE
606B: CD 05 60            CMP  DEPTH      ;IS SHAPE DONE?
606E: 90 D1               BLT  DRAW       ;IF NO, CONTINUE DRAW
6070: 60                  RTS             ;IF YES, STOP
```

```
6071: 03 00 03  59   SHAPE    HEX  030003007D3F01407F3F
6074: 00 7D 3F 01 40 7F 3F
                     HI
                     LO

507 bytes

Symbol table - numerical order:

  LOW     =$1A      HIGH     =$1B      BYTE    =$6003    LINE    =$6004
  DEPTH   =$6005    XCOUNT   =$6006    PGM     =$6007    CLR1    =$601B
  CLR     =$601F    DRAW     =$6041    SHAPE   =$6071    HI      =$607B
  LO      =$613B    GRAPHICS=$C050     MIXOFF  =$C052    PAGE1   =$C054
  HIRES   =$C057
```

Program 3-8 illustrates the general principles of DRAW routines for shapes of any width. For example, here is a routine for a shape 3-bytes wide:

```
DRAW  LDY BYTE
      LDX LINE
      LDA HI,X
      STA HIGH
      LDA LO,X
      STA LOW
      LDX XCOUNT
      LDA SHAPE,X      First shape byte
      STA (LOW),Y      Plot at first screen byte
      INY
      LDA SHAPE+1,X    Second shape byte
      STA (LOW),Y      Plot at second screen byte
      INY
      LDA SHAPE+2,X    Third shape byte
      STA (LOW),Y      Plot at third screen byte
      INC XCOUNT       Increment XCOUNT by shape width
      INC XCOUNT
      INC XCOUNT
      INC LINE
      LDA LINE
      CMP DEPTH
      BLT DRAW
      RTS
```

We now know how to display any shape anywhere on the hi-res screen using shape tables and line address tables. Following chapters will discuss how to move shapes around the screen using animation routines.

4

Vertical Animation

There was a young man named Brown
On whose brow Program 4-2 produced a frown,
"I understand it all right
But there's been an oversight
What goes up is not coming down."

Computer animation is an illusion. Shapes do not move in a continuous, unbroken path but rather in fits and starts, bit by bit (literally!), or sometimes byte by byte. The illusion is created essentially by speed, in the same way that rapidly changing still pictures create the illusion of movement in movie films. We touched on this before in discussing why the speed of assembly language is essential to animation. But speed is not the only factor. The basic cycle for any animation routine is as follows:

Draw → Delay → Erase → Move to new position

If the new position is close to the old one and if the process is fast enough, the illusion of continuous movement is created. The reason for the time delay is to ensure that the shape is on the screen longer than it is off; otherwise, excessive flicker will result.

ERASING A SHAPE

Before we get to the actual vertical animation programs, we first have to discuss the problem of the shape erase. We could erase a shape by clearing the entire screen with our clear screen routine but obviously this would be inappropriate if there are other shapes on the screen we want to retain. We could also just store zeros in the general shape area, but there is an easier and neater way. For this we have to introduce another assembly language instruction, EOR (Exclusive-OR). EOR compares a byte, bit by bit, with a byte in the Accumulator. If either bit, but not both, is one, the result is one: otherwise, the result is zero. The result is stored in the Accumulator.

Example:

Accumulator	1 1 0 1 0 0 1 1
EOR byte	0 1 1 0 1 0 1 0
Result in Accumulator	1 0 1 1 1 0 0 1

Let's see how the EOR instruction can be used to erase a shape. Suppose we load the Accumulator with a shape byte from a particular screen location. Then if we EOR the Accumulator with the same shape byte and store the result at the same screen location, the shape will be erased.

		Content of screen location $NNNN
LDA $NNNN	0 0 1 0 1 1 0 1 (in Accumulator)	#$2D
EOR #$2D	0 0 1 0 1 1 0 1	
Result	0 0 0 0 0 0 0 0 (in Accumulator)	
STA $NNNN		#$00

Pretty neat, eh? But wait, there's more. We can use this same EOR routine not only to erase, but also to draw a shape. All that's necessary is to have a zero stored at the screen location initially.

		Content of screen location $NNNN
LDA $NNNN	0 0 0 0 0 0 0 0 (in Accumulator)	#$00
EOR #$2D	0 0 1 0 1 1 0 1	
Result	0 0 1 0 1 1 0 1 (in Accumulator)	
STA $NNNN		#$2D

This makes life a bit easier for beleaguered assembly language programmers (us), because now we can use a single routine to both draw and erase a shape. The shape is drawn if the screen location contains a zero, and erased if the screen location already contains the shape byte. Alternate calls to the EOR routine will produce a draw-erase cycle. To recapitulate briefly:

Ordinary draw routine	LDA shape byte STA screen location
Draw with EOR	LDA screen location contents (zero) EOR shape byte STA same screen location

Erase with EOR	LDA screen location contents (shape byte) EOR same shape byte STA same screen location

TIME DELAYS

We now need a routine to introduce a time delay in our programs. For this we can take advantage of certain subroutines built into the Apple's operating system (for details, see the Apple Reference Manual). These subroutines perform many functions, from ringing a bell to printing a character. The subroutine we're interested in is at memory location $FCA8. When $FCA8 is accessed, a delay results, the length of which depends on the number in the Accumulator. For example, the following instructions:

```
LDA #$40
JSR  $FCA8 (JSR means Jump to SubRoutine)
```

will produce a delay of approximately 0.01 second. The larger the number in the Accumulator, the longer the delay. In most of our programs, we're going to define the label WAIT as $FCA8 and reserve a memory location for the number to be loaded into the Accumulator; we'll call this DELAY. We then can load DELAY with a number:

```
LDA #$40
STA DELAY
```

A delay is then produced by:

```
LDA DELAY
JSR  WAIT
```

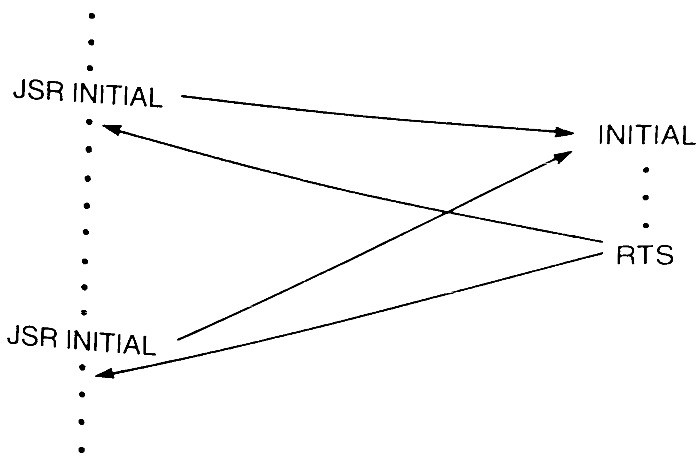
This comes in handy if we want the same delay in several different routines. To see the effect of different delay times, we need change only the value in DELAY. For programs using different delays, we would LDA with the appropriate byte and do a JSR WAIT.

VERTICAL ANIMATION—ONE SHAPE MOVING DOWN

Let's get now to our first vertical animation program. The concept of vertical animation is relatively simple—we draw a shape, delay, erase it, and redraw it either one line down if we're moving down or one line up if we're moving up. We then access the routine repeatedly to traverse the screen (we actually could move any number of lines at a time but a one-line move produces the smoothest results—we're going to use one-line moves for the programs in this chapter).

Our first program (Program 4-1) will move an old friend, the person shape, from the top of the screen to the bottom in a straight line. When it reaches the bottom, the shape will disappear only to reappear at the top for another screen traversal. This will continue ad infinitum until the program is stopped with CONTROL RESET. The program will be using EOR draw routines and also a few things we haven't seen before, so let's discuss some of the details.

First of all, we're going to use the JSR (Jump to SubRoutine) instruction rather extensively. JSR is equivalent to the GOSUB instruction in BASIC. All subroutines called by JSR must end with RTS (ReTurn from Subroutine) in the same way that BASIC subroutines must end with the RETURN instruction.



Although using subroutines does slow a program somewhat (it takes time for any jump instruction), the time lost in most programs is insignificant and is certainly overshadowed by the great advantage, especially for beginners, of providing greater clarity in designing and reading the program. The use of subroutines allows one to divide a program conveniently in two parts—the MAIN PROGRAM and SUBROUTINES. The MAIN PROGRAM gives us an overall view of the program's organization, whereas the SUBROUTINES supply most of the details. In the program we're about to discuss, for example, one can look at the MAIN PROGRAM and take in, almost at a glance, what's going on.

Program 4-1 starts with the usual display and clear screen routines. In addition, #$40 is chosen as the DELAY byte (lines 37 to 38). Let's now look at the MAIN PROGRAM in some detail.

START	JSR	INITIAL	A call to the INITIAL subroutine sets the initial byte position, line number, and depth of the shape.
START1	JSR	DRAW	The shape is drawn with an EOR routine.
	LDA	DELAY	
	JSR	WAIT	A time delay is introduced.
	LDA	LINEA	
	STA	LINE	Because the shape is drawn line-by-line starting from the top and working down, to erase the shape using the same EOR-draw routine, the starting line number for the erase has to be reset to its original value; e.g.,

```
DRAW
----------
Line 0
Line 1
Line 2
Line 3
Line 4
Line 5


ERASE
----------
Line 0
etc.
```

LINEA is used as a repository for the original line number—unlike LINE, it is not changed by the DRAW subroutine.

JSR DRAW — This call to the DRAW subroutine now erases the shape since the screen locations already contain the shape bytes.

```
INC DEPTH
INC LINEA
LDA LINEA
STA LINE
```

Because we're moving the shape down, we want the top of the shape to begin at a new line, one down from the previous position. To move down a line, we increment, as lines are counted 0 to 191, top to bottom. The new line number is stored in LINE and also in LINEA so that it can be recalled for the erase routine. Note that we do not do an INC LINE because LINE has been altered by the DRAW subroutine. DEPTH is also incremented so that the DRAW routine will draw the entire shape.

CMP #$BB — This compares the new line number in the Accumulator to the value #$BB to see if the shape has reached the bottom of the screen. If it has, we want to erase the last drawn shape and start over from the beginning, or at least do something other than allowing the shape to go beyond the screen border. If this happens, the shape may appear in unexpected locations and you will lose control of your program (you could always pull the plug at this point to show who's boss, but let's be more elegant). To see why we selected #$BB as the comparison byte, we should look at how the shape is drawn as it approaches south of the border.

LINE NUMBER Decimal	Hex			
185	#$B9	●		
186	#$BA	● ● ● ● ●	●	
187	#$BB	● ● ● ● ●	● ● ● ● ●	●
188	#$BC	● ● ●	● ● ● ● ●	● ● ● ● ●
189	#$BD	● ●	● ● ●	● ● ● ● ●
190	#$BE	● ●	● ●	● ● ●
191 (bottom)	#$BF		● ●	● ●

The value we want to use in this comparison is the top or starting line of the shape (it doesn't have to be; it's just that we're drawing the shape from top to bottom). Thus, the last shape we want to draw (and erase) starts at line 186 (#$BA). If we start a shape at line 187 (#$BB), part of it will be off the screen.

BGE START

JMP START1 BGE (Branch if Greater or Equal) can be used by some assemblers in place of the normal BCS (Branch if Carry Set). Together with the CMP #$BB instruction, it says that if the number in the Accumulator (the new line number) is greater than or equal to #$BB, branch back to START to begin animation from the initial parameters, i.e., the top of the screen. This branch will be taken when the line number reaches #$BB. If the line number is less than #$BB, the branch will not be taken and the JMP instruction sends the program back to continue drawing from the last line number.

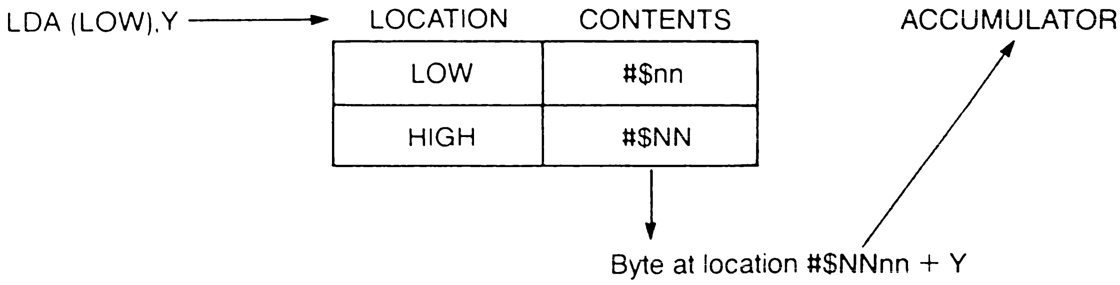
The rule of thumb when using these instructions to test for the bottom of the screen is to subtract the shape depth from 193 (193 − 6 = 187 = #$BB). It really isn't all that complicated once you understand the principles involved (right?).

Finally, note that the last JSR DRAW before this comparison is a shape erase. Thus, we are not left with a shape on the screen when we start again from the top of the screen.

The INITIAL subroutine in this program is essentially self-explanatory. It is here we set the initial line number (0 for top of screen) and the screen byte position—I've chosen #$10 (decimal 16) just to get the shape away from the border.

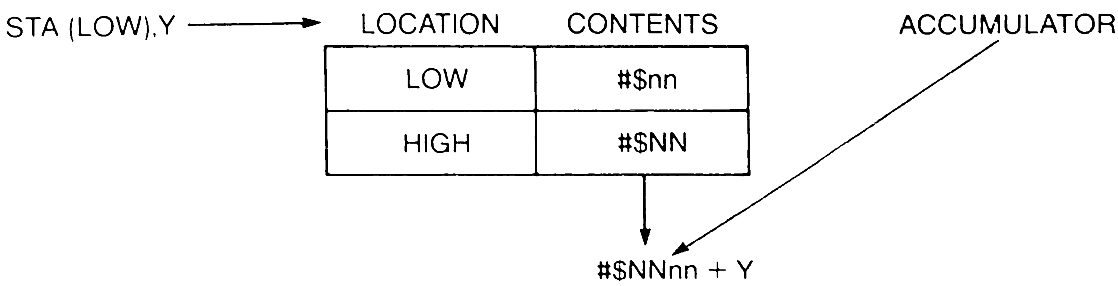
The DRAW subroutine should be familiar to you. We load Y with the byte position, X with the line number, use the HI and LO tables to get line addresses, and XCOUNT for accessing the shape table. We then use an EOR routine for both drawing and erasing.

LDA (LOW),Y Load the Accumulator with the byte at the screen position determined by X and Y.

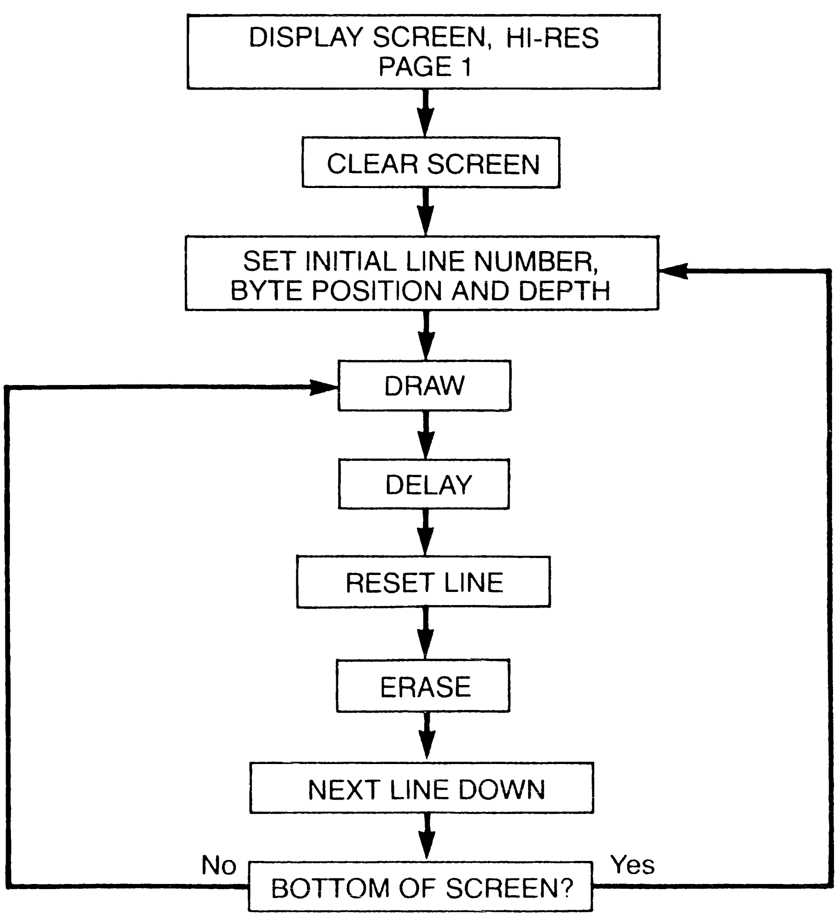


EOR SHAPE,X   EOR the Accumulator with a byte from the shape table (X is loaded from XCOUNT).

STA (LOW),Y   Store the result at the same screen position.



Because the screen initially is clear, when first accessed these instructions will draw. When accessed next, they will erase.



```
]PROGRAM 4-1
:ASM
                  *ONE SHAPE VERTICAL MOVING DOWN
                  ****************************
                  *SHAPE IS 1 BYTE WIDE BY 6 BYTES DEEP
                  ***************************
                            ORG  $6000
6000: 4C 09 60              JMP  PGM
                  XCOUNT    DS   1
                  BYTE      DS   1
                  LINE      DS   1
                  LINEA     DS   1
                  DEPTH     DS   1
                  DELAY     DS   1
                  GRAPHICS  =    $C050
                  MIXOFF    =    $C052
                  HIRES     =    $C057
                  PAGE1     =    $C054
                  HIGH      =    $1B
                  LOW       =    $1A
                  WAIT      =    $FCA8
6009: AD 50 C0    PGM       LDA  GRAPHICS   ;HIRES,P.1
600C: AD 52 C0              LDA  MIXOFF
600F: AD 57 C0              LDA  HIRES
6012: AD 54 C0              LDA  PAGE1
6015: A9 00                 LDA  #$00       ;CLEAR SCREEN 1
6017: 85 1A                 STA  LOW
6019: A9 20                 LDA  #$20
601B: 85 1B                 STA  HIGH
601D: A0 00       CLR1      LDY  #$00
601F: A9 00                 LDA  #$00
6021: 91 1A       CLR       STA  (LOW),Y
6023: C8                    INY
6024: D0 FB                 BNE  CLR
6026: E6 1B                 INC  HIGH
6028: A5 1B                 LDA  HIGH
602A: C9 40                 CMP  #$40
602C: 90 EF                 BLT  CLR1
602E: A9 40                 LDA  #$40       ;LOAD TIME DELAY
6030: 8D 08 60              STA  DELAY
                  ********** MAIN PROGRAM **********
6033: 20 5B 60    START     JSR  INITIAL    ;SETUP BYTE,LINE & DEPTH
6036: 20 6F 60    START1    JSR  DRAW       ;DRAW SHAPE
6039: AD 08 60              LDA  DELAY      ;DELAY
603C: 20 A8 FC              JSR  WAIT
603F: AD 06 60              LDA  LINEA      ;RESET LINE TO
6042: 8D 05 60              STA  LINE            ORIGINAL LINE
6045: 20 6F 60              JSR  DRAW       ;ERASE SHAPE
6048: EE 07 60              INC  DEPTH      ;NEXT DEPTH
604B: EE 06 60              INC  LINEA          & NEXT LINE
604E: AD 06 60              LDA  LINEA
6051: 8D 05 60              STA  LINE
6054: C9 BB                 CMP  #$BB       ;IS LINE AT BOTTOM OF SCREEN?
6056: B0 DB                 BGE  START      ;IF YES, DRAW FROM INITIAL VALUES
6058: 4C 36 60              JMP  START1     ;IF NO, DRAW NEXT LINE
                  ********** SUBROUTINES **********
605B: A9 10       INITIAL   LDA  #$10
605D: 8D 04 60              STA  BYTE       ;SET STARTING BYTE
6060: A9 00                 LDA  #$00
6062: 8D 05 60              STA  LINE       ;SET STARTING LINE
```

```
6065: 8D 06 60              STA   LINEA
6068: 18                    CLC
6069: 69 06                 ADC   #$06          ;ADD DEPTH OF SHAPE TO LINE
606B: 8D 07 60              STA   DEPTH
606E: 60                    RTS
606F: A9 00        DRAW     LDA   #$00
6071: 8D 03 60              STA   XCOUNT        ;ZERO XCOUNT
6074: AC 04 60     DRAW1    LDY   BYTE          ;LOAD BYTE
6077: AE 05 60              LDX   LINE          ;LOAD LINE
607A: BD A3 60              LDA   HI,X          ;LOAD LINE ADDRESS INTO HIGH,LOW
607D: 85 1B                 STA   HIGH
607F: BD 63 61              LDA   LO,X
6082: 85 1A                 STA   LOW
6084: AE 03 60              LDX   XCOUNT        ;LOAD X WITH XCOUNT
6087: B1 1A                 LDA   (LOW),Y       ;GET BYTE FROM SCREEN
6089: 5D 9D 60              EOR   SHAPE,X       ;EOR BYTE FROM SHAPE ADDRESS+X
608C: 91 1A                 STA   (LOW),Y       ;PLOT BYTE
608E: EE 03 60              INC   XCOUNT
6091: EE 05 60              INC   LINE          ;NEXT LINE
6094: AD 05 60              LDA   LINE
6097: CD 07 60              CMP   DEPTH         ;FINISH SHAPE?
609A: 90 D8                 BLT   DRAW1         ;IF NO, DRAW NEXT LINE
609C: 60                    RTS                 ;IF YES, NEXT DRAW CYCLE
609D: 08 3E 5D     SHAPE    HEX   083E5D1C1422 ;SHAPE TABLE
60A0: 1C 14 22
                   HI
                   LO

547 bytes


Symbol table - numerical order:

   LOW     =$1A      HIGH    =$1B      XCOUNT  =$6003     BYTE    =$6004
   LINE    =$6005    LINEA   =$6006    DEPTH   =$6007     DELAY   =$6008
   PGM     =$6009    CLR1    =$601D    CLR     =$6021     START   =$6033
   START1  =$6036    INITIAL =$605B    DRAW    =$606F     DRAW1   =$6074
   SHAPE   =$609D    HI      =$60A3    LO      =$6163     GRAPHICS=$C050
   MIXOFF  =$C052    PAGE1   =$C054    HIRES   =$C057     WAIT    =$FCA8
```

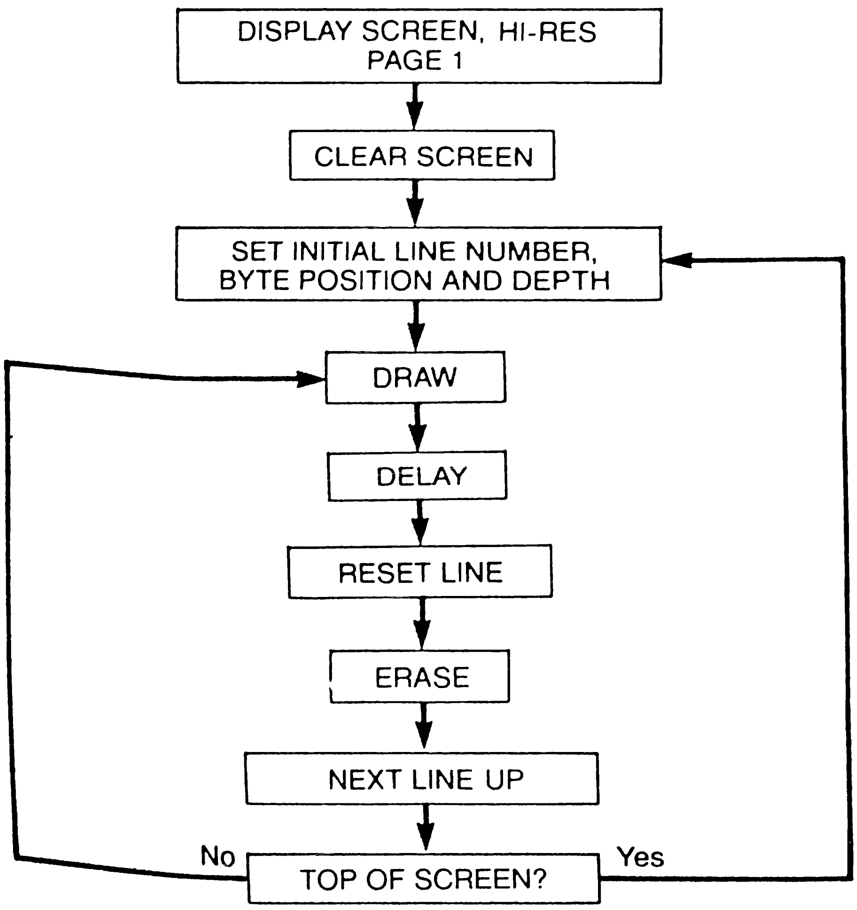
ONE SHAPE MOVING UP

Suppose now we want to move a shape up, from the bottom to the top of the screen (see Program 4-2). There are very few changes that have to be made. First, in the INITIAL subroutine, we set the starting line to #$BA (186). The shape is 6 lines deep, so the first shape will be drawn from lines 186 to 191, just at the bottom border. Second, in the MAIN PROGRAM, we decrement (DEC) LINE and DEPTH instead of increment, because going up means lower line numbers. Then, to test for the top border, we check if LINE has reached zero:

```
LDA LINEA
STA LINE
BEQ START
JMP START1
```

A CMP #$00 is not needed here because BEQ executes a branch if the result of a previous operation was zero. Thus, when LINE equals zero, the branch is taken and the shape begins another journey from the screen bottom. These instructions actually stop (and erase) the shape at line 1. This is of little import in our programs, as a one-line difference at a screen border is hardly noticeable.

Finally, the shape has been changed (easy to do just by changing bytes in the shape table) from a person to a sort of spaceship, as it's a bit disquieting to see a person rising without any visible means of support.




```
]PROGRAM 4-2
:ASM

                    *ONE SHAPE VERTICAL MOVING UP
                    ****************************
                    *SHAPE IS 1 BYTE WIDE BY 6 BYTES DEEP
                    ****************************
                             ORG    $6000
6000: 4C 09 60               JMP    PGM
                    XCOUNT   DS     1
                    BYTE     DS     1
                    LINE     DS     1
                    LINEA    DS     1
                    DEPTH    DS     1
                    DELAY    DS     1
                    GRAPHICS =      $C050
                    MIXOFF   =      $C052
                    HIRES    =      $C057
```

```
                 PAGE1     =      $C054
                 HIGH      =      $1B
                 LOW       =      $1A
                 WAIT      =      $FCA8
6009: AD 50 C0   PGM       LDA    GRAPHICS     ;HIRES,P.1
600C: AD 52 C0             LDA    MIXOFF
600F: AD 57 C0             LDA    HIRES
6012: AD 54 C0             LDA    PAGE1
6015: A9 00                LDA    #$00         ;CLEAR SCREEN 1
6017: 85 1A                STA    LOW
6019: A9 20                LDA    #$20
601B: 85 1B                STA    HIGH
601D: A0 00      CLR1      LDY    #$00
601F: A9 00                LDA    #$00
6021: 91 1A      CLR       STA    (LOW),Y
6023: C8                   INY
6024: D0 FB                BNE    CLR
6026: E6 1B                INC    HIGH
6028: A5 1B                LDA    HIGH
602A: C9 40                CMP    #$40
602C: 90 EF                BLT    CLR1
602E: A9 40                LDA    #$40         ;LOAD TIME DELAY
6030: 8D 08 60             STA    DELAY
                 ********** MAIN PROGRAM **********
6033: 20 59 60   START     JSR    INITIAL      ;SETUP BYTE,LINE & DEPTH
6036: 20 6D 60   START1    JSR    DRAW         ;DRAW SHAPE
6039: AD 08 60             LDA    DELAY        ;DELAY
603C: 20 A8 FC             JSR    WAIT
603F: AD 06 60             LDA    LINEA        ;RESET LINE TO
6042: 8D 05 60             STA    LINE               ORIGINAL LINE
6045: 20 6D 60             JSR    DRAW         ;ERASE SHAPE
6048: CE 07 60             DEC    DEPTH        ;NEXT DEPTH
604B: CE 06 60             DEC    LINEA              & NEXT LINE
604E: AD 06 60             LDA    LINEA
6051: 8D 05 60             STA    LINE         ;IS LINE AT TOP OF SCREEN?
6054: F0 DD                BEQ    START        ;IF YES, DRAW FROM INITIAL VALUES
6056: 4C 36 60             JMP    START1       ;IF NO, DRAW NEXT LINE
                 ********** SUBROUTINES **********
6059: A9 10      INITIAL   LDA    #$10
605B: 8D 04 60             STA    BYTE         ;SET STARTING BYTE
605E: A9 BA                LDA    #$BA
6060: 8D 05 60             STA    LINE         ;SET STARTING LINE
6063: 8D 06 60             STA    LINEA
6066: 18                   CLC
6067: 69 06                ADC    #$06         ;ADD DEPTH OF SHAPE TO LINE
6069: 8D 07 60             STA    DEPTH
606C: 60                   RTS
606D: A9 00      DRAW      LDA    #$00
606F: 8D 03 60             STA    XCOUNT       ;ZERO XCOUNT
6072: AC 04 60   DRAW1     LDY    BYTE         ;LOAD BYTE
6075: AE 05 60             LDX    LINE         ;LOAD LINE
6078: BD A1 60             LDA    HI,X         ;LOAD LINE ADDRESS INTO HIGH,LOW
607B: 85 1B                STA    HIGH
607D: BD 61 61             LDA    LO,X
6080: 85 1A                STA    LOW
6082: AE 03 60             LDX    XCOUNT       ;LOAD X WITH XCOUNT
6085: B1 1A                LDA    (LOW),Y      ;GET BYTE FROM SCREEN
```

```
6087: 5D 9B 60            EOR  SHAPE,X       ;EOR BYTE FROM SHAPE ADDRESS+X
608A: 91 1A               STA  (LOW),Y       ;PLOT BYTE
608C: EE 03 60            INC  XCOUNT
608F: EE 05 60            INC  LINE          ;NEXT LINE
6092: AD 05 60            LDA  LINE
6095: CD 07 60            CMP  DEPTH         ;FINISH SHAPE?
6098: 90 D8               BLT  DRAW1         ;IF NO, DRAW NEXT LINE
609A: 60                  RTS                ;IF YES, NEXT DRAW CYCLE
609B: 08 1C 22    SHAPE   HEX  081C223E227F ;SHAPE TABLE
609E: 3E 22 7F
                  HI
                  LO

545 bytes

Symbol table - numerical order:

   LOW      =$1A      HIGH     =$1B      XCOUNT   =$6003    BYTE      =$6004
   LINE     =$6005    LINEA    =$6006    DEPTH    =$6007    DELAY     =$6008
   PGM      =$6009    CLR1     =$601D    CLR      =$6021    START     =$6033
   START1   =$6036    INITIAL  =$6059    DRAW     =$606D    DRAW1     =$6072
   SHAPE    =$609B    HI       =$60A1    LO       =$6161    GRAPHICS=$C050
   MIXOFF   =$C052    PAGE1    =$C054    HIRES    =$C057    WAIT      =$FCA8
```


DRAW-DRAW ROUTINES

We've seen, in the previous two programs, how to erase a shape using the EOR instruction. Let's call this type of routine DRAW-ERASE. There is yet another way to erase a shape and that is by drawing over it, a process that has advantages as well as disadvantages. We'll call this type of routine DRAW-DRAW. The salient point here is that when a byte is sent to a screen position, the byte (if any) already present at that position is replaced by the new byte.

Example

	Contents of screen location $NNNN
	#$17
LDA #$23 STA $NNNN	
	#$23

Let's adapt the DRAW-DRAW routine to Program 4-1, moving a shape down the screen (see Program 4-3). The shape is drawn with an ordinary draw (LDA shape byte, STA screen location) instead of the EOR routine. The shape is moved down one line at a time without any erase routine. Let's follow the shape moving down two lines.

LINE		NEXT LINE DOWN	NEXT LINE DOWN
0	●	●	●
1	●●●●●	●	●
2	● ●●● ●	●●●●●	●
3	●●●	● ●●● ●	●●●●●
4	● ●	●●●	● ●●● ●
5	● ●	● ●	●●●
6		● ●	● ●
7			● ●

As you can see, each shape byte, as it moves down one line, erases the byte that was there before, thus preserving the shape. As you can also see, something's not quite right. We're always left with the top byte on the screen, because nothing moves into those positions. We solve this problem by providing the shape with a border of #$00 at the top. Now see what happens.

LINE		NEXT LINE DOWN	NEXT LINE DOWN
0	#$00		
1	●	#$00	
2	●●●●●	●	#$00
3	● ●●● ●	●●●●●	●
4	●●●	● ●●● ●	●●●●●
5	● ●	●●●	● ●●● ●
6	● ●	● ●	●●●
7		● ●	● ●
8			● ●

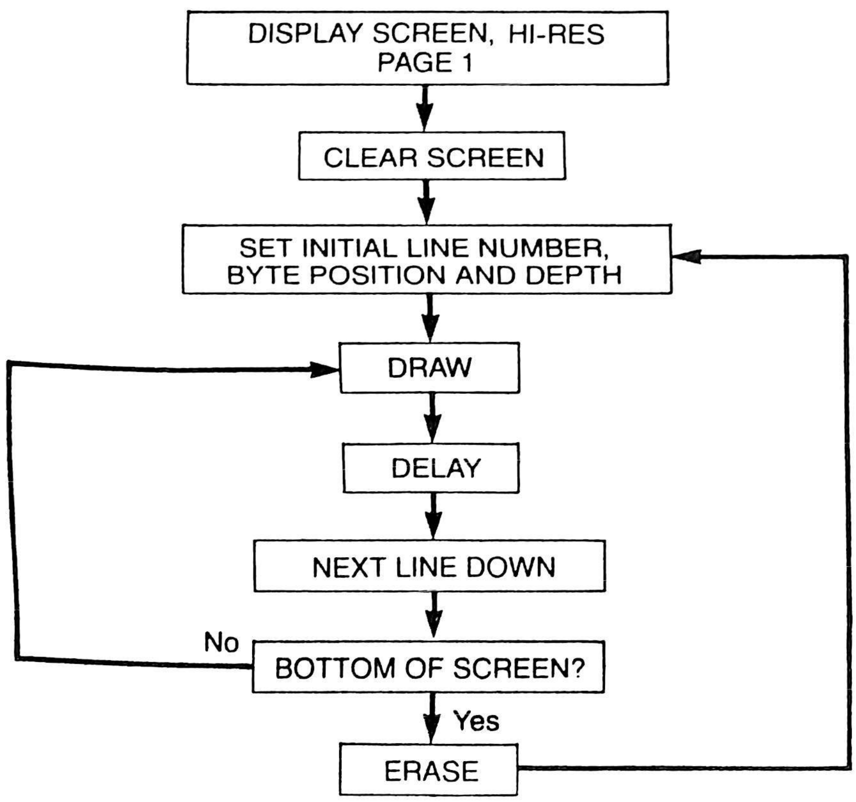
The border is always placed behind the direction of movement and serves to erase the first line of the shape. To introduce the border into the person shape, we add #$00 at the beginning of the shape table. Thus, the person-shape table with a trailing border is 00083E5D1C1422 (compare to Program 4-1). We must also remember to change the shape depth from 6 to 7 in the INITIAL subroutine. A general rule is that the border size has to equal the maximum shape move. Thus, if we're moving a shape two lines at a time, the trailing border would be two #$00's.

There is one further complication we have to deal with in programs that use DRAW-DRAW routines. For example, in the program we're discussing, when the shape reaches the bottom of the screen, it will stop and then appear again from the top. Then, because we have no erase instructions, the shape at the bottom stays on the screen. We have to introduce an erase routine to erase the last shape when it reaches a border. For this, we can use our usual EOR instructions in a routine called ERASE. Thus, in the MAIN PROGRAM of Program 4-3,

CMP #$BA — #$BA is used instead of #$BB as in Program 4-1 because this shape is 7 lines deep due to the border (193 − 7 = 186 = #$BA).

BGE ERASE

JMP START1 — Now the comparison tells us if the shape is at the screen bottom, go to the ERASE routine, which erases the shape and then sends the program back to START to continue the animation from the initial parameters, i.e., top of the screen.




```
]PROGRAM 4-3
:ASM
                *ONE SHAPE VERTICAL MOVING DOWN; DRAW-DRAW CYCLE
                ****************************
                *SHAPE IS 1 BYTE WIDE BY 7 BYTES DEEP
                ****************************
                         ORG   $6000
6000: 4C 09 60           JMP   PGM
                XCOUNT   DS    1
                BYTE     DS    1
                LINE     DS    1
                LINEA    DS    1
                DEPTH    DS    1
                DELAY    DS    1
                GRAPHICS =     $C050
                MIXOFF   =     $C052
                HIRES    =     $C057
                PAGE1    =     $C054
```

```
                   HIGH     =     $1B
                   LOW      =     $1A
                   WAIT     =     $FCA8
6009: AD 50 C0     PGM      LDA   GRAPHICS     ;HIRES,P.1
600C: AD 52 C0              LDA   MIXOFF
600F: AD 57 C0              LDA   HIRES
6012: AD 54 C0              LDA   PAGE1
6015: A9 00                 LDA   #$00         ;CLEAR SCREEN 1
6017: 85 1A                 STA   LOW
6019: A9 20                 LDA   #$20
601B: 85 1B                 STA   HIGH
601D: A0 00        CLR1     LDY   #$00
601F: A9 00                 LDA   #$00
6021: 91 1A        CLR      STA   (LOW),Y
6023: C8                    INY
6024: D0 FB                 BNE   CLR
6026: E6 1B                 INC   HIGH
6028: A5 1B                 LDA   HIGH
602A: C9 40                 CMP   #$40
602C: 90 EF                 BLT   CLR1
602E: A9 40                 LDA   #$40         ;LOAD TIME DELAY
6030: 8D 08 60              STA   DELAY
                   ********** MAIN PROGRAM **********
6033: 20 52 60     START    JSR   INITIAL      ;SETUP BYTE,LINE & DEPTH
6036: 20 66 60     START1   JSR   DRAW         ;DRAW SHAPE
6039: AD 08 60              LDA   DELAY        ;DELAY
603C: 20 A8 FC              JSR   WAIT
603F: EE 07 60              INC   DEPTH        ;NEXT DEPTH
6042: EE 06 60              INC   LINEA        ;     & NEXT LINE
6045: AD 06 60              LDA   LINEA
6048: 8D 05 60              STA   LINE
604B: C9 BA                 CMP   #$BA         ;IS LINE AT BOTTOM OF SCREEN?
604D: B0 43                 BGE   ERASE        ;IF YES, ERASE SHAPE, START OVER
604F: 4C 36 60              JMP   START1       ;IF NO, DRAW NEXT LINE
                   ********** SUBROUTINES **********
6052: A9 10        INITIAL  LDA   #$10
6054: 8D 04 60              STA   BYTE         ;SET STARTING BYTE
6057: A9 00                 LDA   #$00
6059: 8D 05 60              STA   LINE         ;SET STARTING LINE
605C: 8D 06 60              STA   LINEA
605F: 18                    CLC
6060: 69 07                 ADC   #$07         ;ADD DEPTH OF SHAPE TO LINE
6062: 8D 07 60              STA   DEPTH
6065: 60                    RTS
6066: A9 00        DRAW     LDA   #$00
6068: 8D 03 60              STA   XCOUNT       ;ZERO XCOUNT
606B: AC 04 60     DRAW1    LDY   BYTE         ;LOAD BYTE
606E: AE 05 60              LDX   LINE         ;LOAD LINE
6071: BD CC 60              LDA   HI,X         ;LOAD LINE ADDRESS INTO HIGH,LOW
6074: 85 1B                 STA   HIGH
6076: BD 8C 61              LDA   LO,X
6079: 85 1A                 STA   LOW
607B: AE 03 60              LDX   XCOUNT       ;LOAD X WITH XCOUNT
607E: BD C5 60              LDA   SHAPE,X      ;LOAD SHAPE BYTE
6081: 91 1A                 STA   (LOW),Y      ;PLOT BYTE
6083: EE 03 60              INC   XCOUNT
6086: EE 05 60              INC   LINE         ;NEXT LINE
6089: AD 05 60              LDA   LINE
608C: CD 07 60              CMP   DEPTH        ;FINISH SHAPE?
608F: 90 DA                 BLT   DRAW1        ;IF NO, DRAW NEXT LINE
6091: 60                    RTS                ;IF YES, NEXT DRAW CYCLE
```

```
6092: CE 05 60     ERASE    DEC  LINE
6095: A9 00                 LDA  #$00         ;ZERO XCOUNT
6097: 8D 03 60              STA  XCOUNT
609A: AC 04 60     ERASE1   LDY  BYTE
609D: AE 05 60              LDX  LINE
60A0: BD CC 60              LDA  HI,X
60A3: 85 1B                 STA  HIGH
60A5: BD 8C 61              LDA  LO,X
60A8: 85 1A                 STA  LOW
60AA: AE 03 60              LDX  XCOUNT
60AD: B1 1A                 LDA  (LOW),Y
60AF: 5D C5 60              EOR  SHAPE,X
60B2: 91 1A                 STA  (LOW),Y      ;ERASE
60B4: EE 03 60              INC  XCOUNT
60B7: EE 05 60              INC  LINE
60BA: AD 05 60              LDA  LINE
60BD: CD 07 60              CMP  DEPTH
60C0: 90 D8                 BLT  ERASE1
60C2: 4C 33 60              JMP  START
60C5: 00 08 3E     SHAPE    HEX  00083E5D1C1422 ;SHAPE TABLE
60C8: 5D 1C 14 22
                   HI
                   LO

588 bytes

Symbol table - numerical order:

   LOW     =$1A      HIGH     =$1B      XCOUNT  =$6003    BYTE    =$6004
   LINE    =$6005    LINEA    =$6006    DEPTH   =$6007    DELAY   =$6008
   PGM     =$6009    CLR1     =$601D    CLR     =$6021    START   =$6033
   START1  =$6036    INITIAL  =$6052    DRAW    =$6066    DRAW1   =$606B
   ERASE   =$6092    ERASE1   =$609A    SHAPE   =$60C5    HI      =$60CC
   LO      =$618C    GRAPHICS=$C050     MIXOFF  =$C052    PAGE1   =$C054
   HIRES   =$C057    WAIT     =$FCA8
```


We mentioned before that DRAW-DRAW routines have certain advantages and disadvantages. As there is no erase cycle as such, shapes animated by DRAW-DRAW move faster and with essentially no flicker. (It should be pointed out that "flicker" is a subjective term and depends to some extent on the image retention characteristics of the monitor or TV you're using—long retention times minimize flicker, whereas short times emphasize it, and retention times vary greatly among different brands of display screens.) On the other hand, DRAW-DRAW requires two routines, one to draw and one to erase the last shape (unless, of course, a shape is to stay on the screen). Also, collision detection is difficult with DRAW-DRAW routines (but only with horizontally moving shapes as we'll see in Chapter 7).

In addition, the speed advantage of DRAW-DRAW, at least in simple programs, is more theoretical than practical. If you compare Programs 4-1 and 4-3, you'll see that the shape traverses the screen at about the same speed in both cases. This is because the determining factor is the time delay, which is #$40 in both programs. So while the speed of DRAW-DRAW is greater than DRAW-ERASE, the speed differential is much less than the time delay. The speed advantage of DRAW-DRAW becomes important only in programs with larger and more complicated shapes where drawing and erasing the shape takes up an appreciable amount of time. It should also be noted that a time delay in DRAW-DRAW routines is not necessary to reduce flicker by ensuring that the shape is on the screen longer than it is off because the shape is not erased. However, delays are still generally required to slow a program down to a reasonable pace.

One further drawback of DRAW-DRAW is that it is inappropriate for drawing over backgrounds—this will be discussed in more detail in Chapter 14.

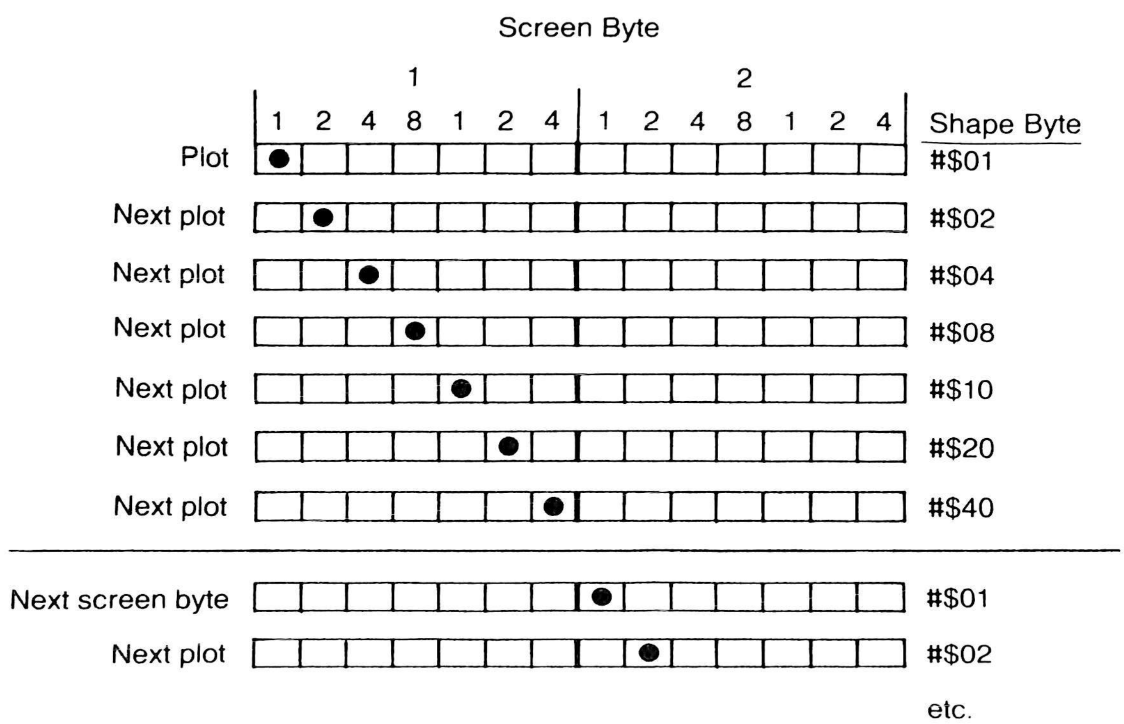
The decision whether to use DRAW-DRAW or DRAW-ERASE routines depends on the particular requirements of the program. If the shape is not involved in collision detection, if you're not drawing over a background, and if more speed and the absence of flicker are desirable, use DRAW-DRAW. If speed and flicker are not problems and collision detection (for shapes moving horizontally) is required, use DRAW-ERASE. The final game program uses DRAW-ERASE routines, mostly because it makes the program easier to write and read, requiring only one draw routine, and speed and flicker are not problems. This should not be construed in any way as relegating DRAW-DRAW routines to second-class status. They are quite useful for smooth and rapid animation and should be kept in mind for your own programs, and indeed for the game program itself; in fact the reader may find it a useful and instructive exercise to modify parts of the final program to DRAW-DRAW. With this in mind, I've included, in later chapters, some routines in both DRAW-ERASE and DRAW-DRAW modes. There will be more on program modifications in the last chapter.

5

Horizontal Movement and Internal Animation

Moving a shape horizontal
Can cause problems periodontal.
The frustrations underneath
Lead to gnashing of teeth
Side to side and back to frontal.

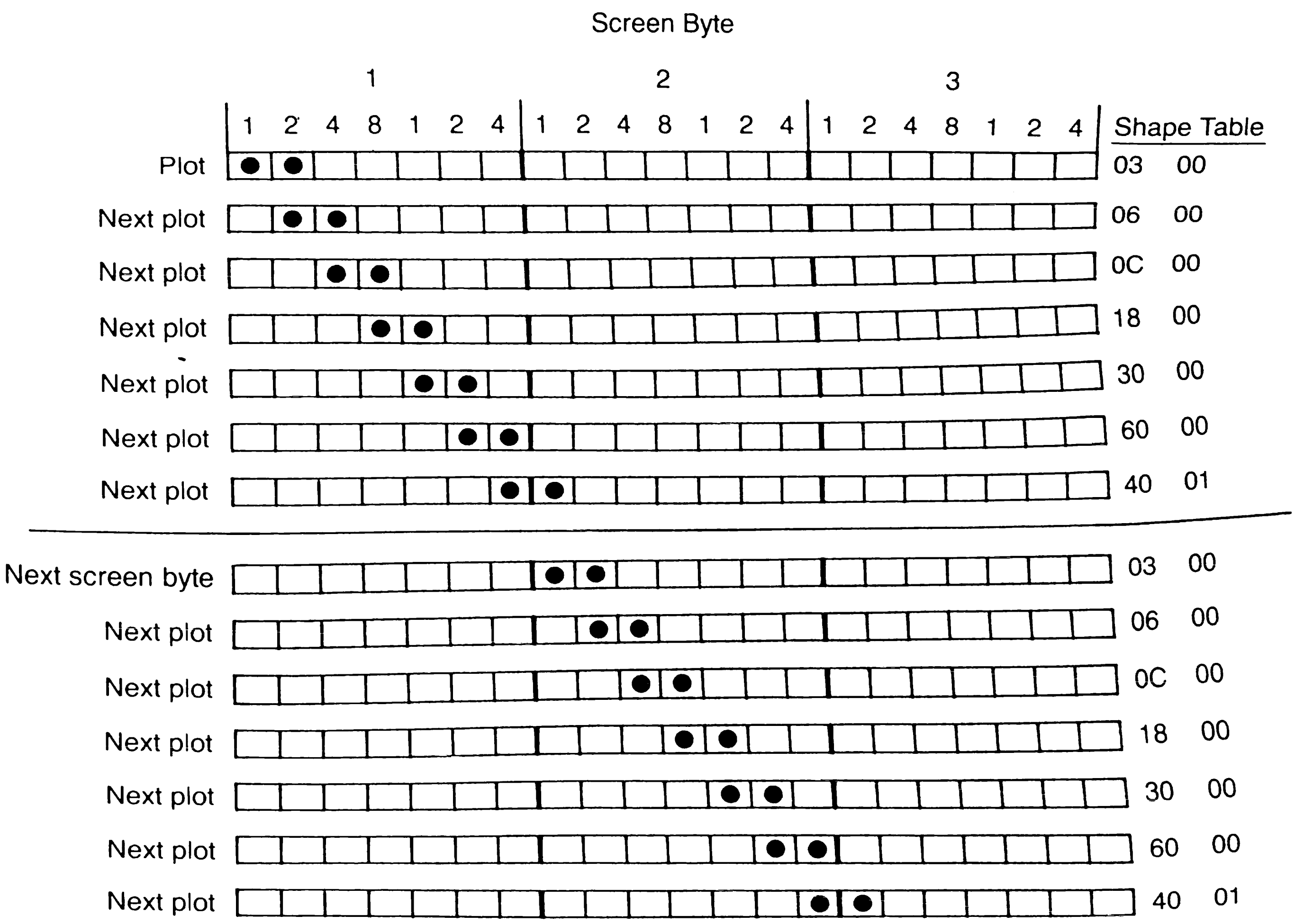
Moving a shape horizontally across the hi-res screen involves the same basic animation principles as vertical movement, i.e., DRAW-DELAY-ERASE-MOVE-DRAW, but a certain complication arises that will become immediately apparent upon examining the following diagram.



THE SEVEN PRESHIFTED SHAPES

Here the shape is a single dot, moving left to right one bit position at a time (we can move the shape any number of bits at a time, but a one-bit move produces the smoothest animation). Obviously what's happening is that every time we move the shape over one bit, the shape byte changes. After moving seven bit positions (one screen byte), the same series of shape bytes is plotted, but now in the next screen byte. Thus, for each shape to be moved horizontally, we need seven different shape bytes (or shape tables in the case of larger shapes). Shapes plotted in this manner are called preshifted shapes. Note that movement results from plotting the different shape bytes and not by changing screen byte positions (except at the screen byte boundaries).

Actually, the example just shown is a special case (one dot at the leftmost position). Let's look at a more general example.



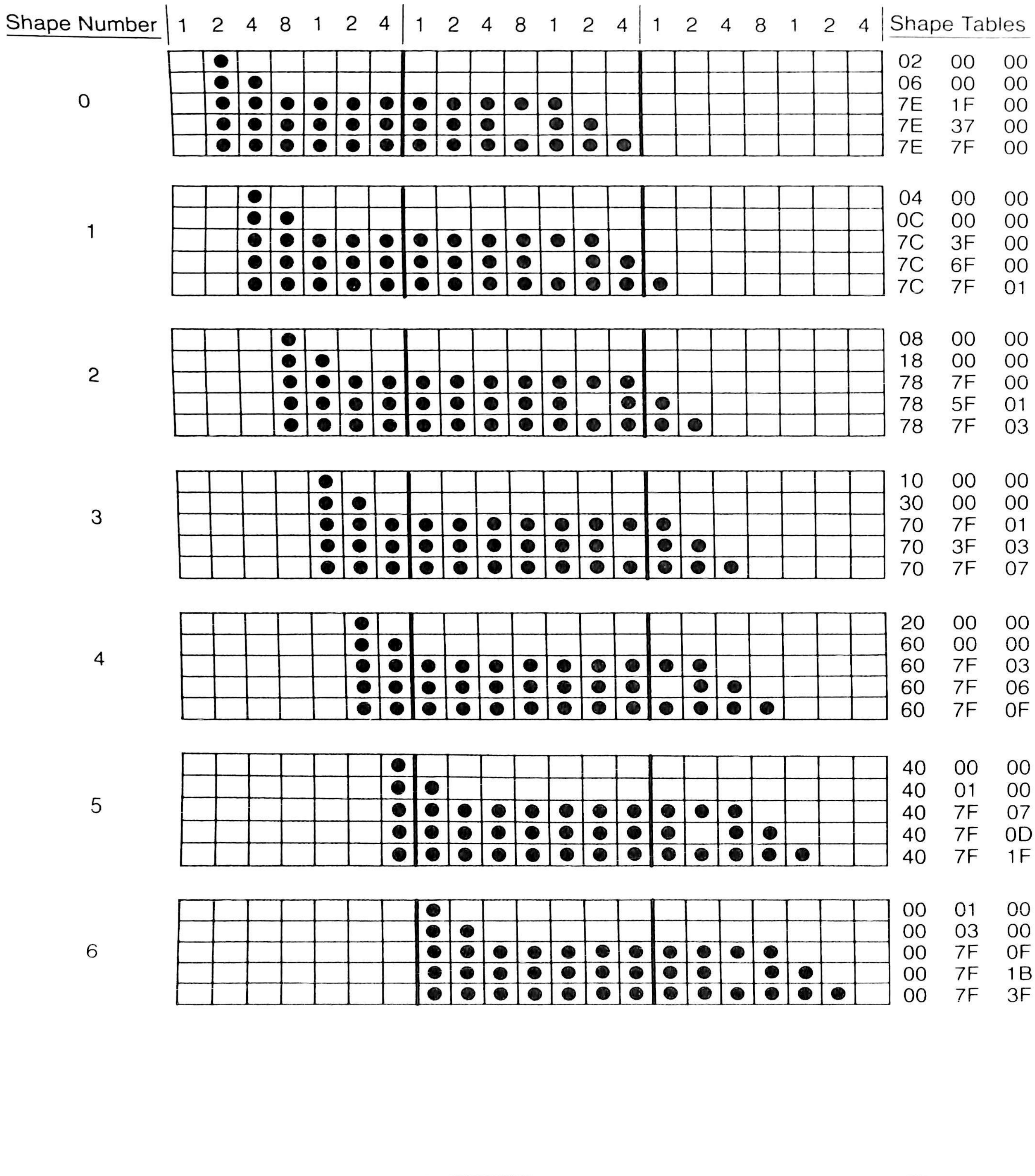
Here we're moving a two-dot shape left to right one bit position at a time. Again, seven different shape tables are required. We also see that the seventh shape is partly in one screen byte and partly in the next. Therefore, in constructing our shape tables, we have to include an extra screen byte in the direction of movement (for one-bit moves). Thus, for a shape one screen byte wide or less, as in the above example, the shape table will cover two screen bytes, and a two screen byte wide shape will require a shape table covering three screen bytes, and so on. This is a general rule, applicable in all cases except the special case of the one-dot shape in the first example.

To summarize, horizontal movement for one-bit moves requires:

1. Seven shape tables for each shape.
2. Shape tables with an extra screen byte in the direction of movement.

Obviously a higher level of complication has been introduced compared to vertical animation, but that's the way it is. There's no way to get around it unless we want to move a shape just one screen byte at a time. In some cases this may be satisfactory, but usually the large distances involved produce an unacceptably jumpy animation.

Let's now look at some actual shape tables we're going to use in our game program. The following diagrams illustrate the seven shape tables for a two-



Shape Number	Shape Tables		
0	02	00	00
	06	00	00
	7E	1F	00
	7E	37	00
	7E	7F	00
1	04	00	00
	0C	00	00
	7C	3F	00
	7C	6F	00
	7C	7F	01
2	08	00	00
	18	00	00
	78	7F	00
	78	5F	01
	78	7F	03
3	10	00	00
	30	00	00
	70	7F	01
	70	3F	03
	70	7F	07
4	20	00	00
	60	00	00
	60	7F	03
	60	7F	06
	60	7F	0F
5	40	00	00
	40	01	00
	40	7F	07
	40	7F	0D
	40	7F	1F
6	00	01	00
	00	03	00
	00	7F	0F
	00	7F	1B
	00	7F	3F

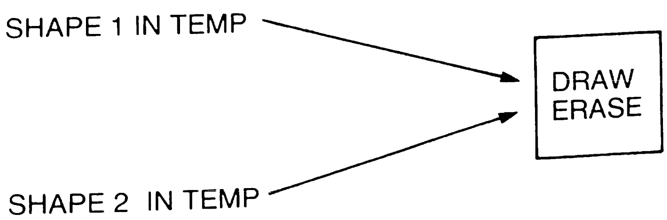
screen-byte-wide airplane that's going to move horizontally across the screen, left to right, one bit position at a time (the tables are labelled 0 to 6 instead of 1 to 7, because they will be referred to by these numbers in the program to facilitate routines that access them).

The trick to incorporating these shapes into a program is to direct each of the seven shapes to its proper location. The following program (5-1) illustrates one approach to this problem. The program moves the plane shape across the screen from left to right. When the shape reaches the right border, it disappears and then reappears at the left border for another screen journey ad infinitum. Thus, we're also going to discuss in this program tests for the vertical ends of screens. Pay attention to Program 5-1 because we're going to use its routines in our final game program.

TEMP AND SHAPE ADDRESS TABLES

A programming technique I generally strive for is to minimize the number of drawing routines as much as possible. This produces a more compact program, easier to write and understand. Program 5-1 has only a single draw routine for all seven preshifted shapes, and both draws and erases using the EOR instruction.

One way to use a single draw routine accessing seven different shape tables is to store the shape bytes temporarily in an area of memory we'll call TEMP (aren't we clever with our labels), and use TEMP instead of the shape tables in the draw routine. To draw any of the seven shapes, we load TEMP with the appropriate shape bytes and access the draw routine. Another advantage in using TEMP is that for the shape erase, TEMP doesn't have to be reloaded because it already contains the appropriate shape bytes, i.e.,



To load the shape bytes into TEMP, the program has to know where in memory the shape tables are located. To do this, we construct a shape address table and let's label it SHPADR (more clever labelling). This table will store the beginning memory locations of each of the seven shape tables. For example, in Program 5-1, the SHAPE1 table begins at location $60F5, SHAPE2 at $6104, SHAPE3 at $6113, and so on. The SHPADR table will look like this:

```
SHPADR  F5    (SHAPE1)
        60
        04    (SHAPE2)
        61
        13    (SHAPE3)
        61
        etc.
```

Shape address tables contain 14 bytes, 2 for each address of the 7 shape tables (note that the shape table addresses are stored low byte first). Now we can access each shape table by referring only to SHPADR—this allows us to use a single routine for loading all the shape tables into TEMP. SHPADR and SHPADR+1 will give us the address of SHAPE1, SHPADR+2 and SHPADR+3 will give us the address of SHAPE2, SHPADR+4 and SHPADR+5 will give us the address of SHAPE3, and so on. More specifics about this technique will be discussed below.

For now we have to discuss how the SHPADR table is constructed. This depends on the type of assembler you're using. Full-feature assemblers support instructions that allow the assembler to construct a shape address table directly from within the program. This is illustrated in Program 5-1. Look at the SHPADR table starting at line 22. The instruction DFB #<SHAPE1 loads the low byte of the SHAPE1 table; DFB #>SHAPE1 loads the high byte (the DOS Tool Kit assembler does this backwards—#> for the low byte and #< for the high byte—the ORCA/M assembler uses the instruction DC A 'shape table'—a good reason to read your assembler's instructions!). The entire SHPADR table is constructed by the assembler using these DFB instructions for all seven shape tables. If your assembler doesn't have this capability, you have a problem, but one that is not insurmountable, merely inconvenient. In this situation, the problem is you (and the assembler) don't know the shape table memory addresses until after the program is assembled, because assemblers simply start at the ORG and then fill up memory sequentially. The solution in this case is to assemble the program without a SHPADR table, write down the memory addresses of the shape tables, and use the edit feature of the assembler to add the SHPADR table at the end of the program. It doesn't make any difference where in the program the SHPADR table is located, as it is accessed by reference to its label and not to a specific memory address.

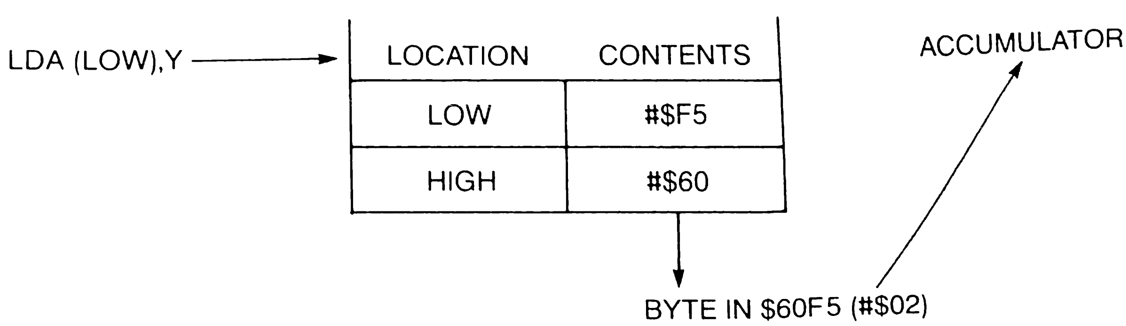
Now that we've constructed the SHPADR table, let's see how it's used to load TEMP with shape bytes. First we have to reserve an area of memory for TEMP. To do this we have to know the total number of bytes in each shape table. The plane shape is 2 screen bytes wide by 5 lines deep. Following the general rule discussed before, the shape table will cover 3 screen bytes. Thus, each shape table will contain 15 bytes (3 × 5). Space for TEMP then is reserved by the instruction TEMP DS 15 (line 12).

Before we load TEMP with shape bytes, we have to specify which of the seven shape tables we're going to use. Let's start the screen traversal at the left border with SHAPE1. In the MAIN PROGRAM section of Program 5-1, the shape number is loaded into a reserved memory location I've labeled SHPNO. For reasons that will soon become clear, SHAPE1 is selected by loading SHPNO with 0 (lines 57 and 58). The next line sends the program to the LOADSHP subroutine—it is here TEMP is loaded with the shape bytes. First, the Accumulator is loaded with the value in SHPNO (LDA SHPNO, line 83). At this point, the value is 0. The next instruction, ASL, is a mnemonic for Arithmetic Shift Left (if the contents of the Accumulator are to be shifted, some assemblers require an A as the operand—the BIG MAC does not). What this does is move each bit in the Accumulator one position to the left—the result is to multiply the number in the Accumulator by 2, i.e.,

128	*64*	*32*	*16*	*8*	*4*	*2*	*1*	*Decimal*
0	0	0	0	0	1	1	0	6
0	0	0	0	1	1	0	0	12
0	0	0	1	1	0	0	0	24

The result of the ASL instruction is stored in the Accumulator. Because the Accumulator contained 0, the result is still 0. The next instruction (TAX—Transfer Accumulator to X-Register) does what it says—the number in the Accumulator is transferred to the X register. X now contains 0. Now the instruction LDA SHPADR,X loads the Accumulator with the byte found at address SHPADR + X; because X = 0, A is loaded with the value #$F5, the byte at location SHPADR. This byte is stored in a zero page location, LOW or $1A. The Accumulator is then loaded with the next byte in the SHPADR table, #$60, by the instruction LDA SHPADR+1,X. This byte is stored in another zero page location, HIGH or $1B. LOW and HIGH now contain the low byte and high byte respectively, of the address of SHAPE1 ($60F5). This completes the process of selecting which shape table is to be loaded into TEMP. The next step is to load TEMP with the shape bytes.

The Y register is loaded with 0 (line 90). The next instruction on line 91 (LDA(LOW),Y) is one we've seen before—indirect indexing. It says load the Accumulator with the byte to be found at a memory address calculated as follows—get the low byte of the address from LOW, the high byte from HIGH, and add the contents of the Y register. The byte found at this address is then loaded into the Accumulator.



The Accumulator now contains the first byte of the SHAPE1 table. The next instruction, STA TEMP,Y stores this byte in the first position of TEMP. The Y register is then incremented by one (INY) and, if it is less than the number of bytes in the shape table (15 or #$0F), CPY #$0F (compare Y to #$0F) and BLT (Branch if Less Than) LOADSHP1 sends the program back to LDA (LOW),Y (line 91) to load the second byte of SHAPE1 into the second position of TEMP, i.e.,

$60F5 + 0; 1st byte in SHAPE1 loaded into 1st position of TEMP
$60F5 + 1; 2nd byte in SHAPE1 loaded into 2nd position of TEMP
$60F5 + 2; 3rd byte in SHAPE1 loaded into 3rd position of TEMP

.
.
.

$60F5 + 14; 15th (last) byte in SHAPE1 loaded into last position of TEMP
$60F5 + 15; stop and return to MAIN PROGRAM

The shape in TEMP is then drawn and erased with the EOR routine we've seen before, except TEMP instead of a shape table is accessed to obtain the shape bytes. We'll discuss the draw routine in more detail below.

Now we would like to draw the next shape, SHAPE2. To do this we increment SHPNO by one (line 64) so that SHPNO now contains the value 1. The

LOADSHP subroutine (line 83) multiplies this by 2 (result = 2), and the result is then placed in the X register (line 85). The instruction LDA SHPADR,X (line 86) now loads the Accumulator with the third byte of the SHPADR table (SHPADR + 2), which is the low byte of the address of SHAPE2 (#$04). This byte is stored in LOW. LDA SHPADR+1,X loads the Accumulator with the fourth byte of SHPADR, which is the high byte of the address of SHAPE2 (#$61). This byte is stored in HIGH. Thus, LOW and HIGH now contain the low and high bytes, respectively, of the address of SHAPE2. The subsequent instructions load the bytes from SHAPE2 into TEMP in preparation for drawing and erasing. In the same way, SHAPE3 is selected by loading SHPNO with 2, SHAPE4 by loading SHPNO with 3, and so on, i.e.,

SHPNO	*ASL*	*X*	*LDA SHPADR,X*	*LDA SHPADR+1,X*	*SHAPE TABLE*
0	0	0	SHPADR + 0	SHPADR + 1	1
1	2	2	SHPADR + 2	SHPADR + 3	2
2	4	4	SHPADR + 4	SHPADR + 5	3
3	6	6	SHPADR + 6	SHPADR + 7	4
4	8	8	SHPADR + 8	SHPADR + 9	5
5	10	10	SHPADR + 10	SHPADR + 11	6
6	12	12	SHPADR + 12	SHPADR + 13	7

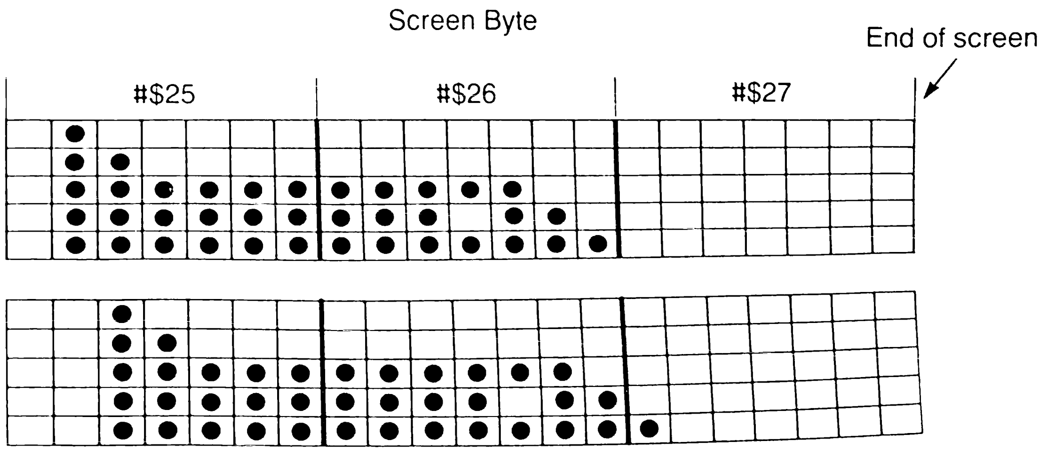
ACCESSING SEQUENTIAL SHAPES AND TESTING FOR END OF SCREEN

Let's look at the MAIN PROGRAM section of Program 5-1. The first instruction accesses the INITIAL subroutine, which sets the initial screen byte and line (0 in both cases) and also specifies the shape depth. Because we want to start with SHAPE1, SHPNO is loaded with 0. We then load TEMP, draw, delay, and erase. The erase is accomplished by the DRAW routine itself because we're using the EOR (DRAW-ERASE) technique discussed in previous chapters. Note that TEMP already contains the bytes of the shape we want to erase and so TEMP does not have to be reloaded with shape bytes for the erase routine.

We're now ready to draw and erase SHAPE2. To do this we first load SHPNO with 1 by INC SHPNO. The next instructions (LDA SHPNO, CMP #$07, BLT START2) say if the value in SHPNO is less than 7, continue the program at START2; i.e., load TEMP, draw, delay, erase. SHPNO is incremented again for the next shape and so on until SHPNO contains the value 7. At this point, we've plotted the seven shapes (0 to 6 in SHPNO) in the first screen byte.

We now want to start over with SHAPE1 but at the next screen byte. Because SHPNO contains the value 7, the branch BLT START2 (line 67) is not taken and the program skips to the next line (INC BYTE), which increments BYTE by 1. The DRAW routine will now draw in the next screen byte. Before we draw, however, we have to test to see if the shape has reached the right end of the screen because we can't allow the shape to go beyond the screen boundaries. To do this, we load the Accumulator with the value in BYTE and compare it to the value #$26 (decimal 38). If the value is less than #$26, the branch in line 71 (BLT START1) is taken and the program continues with all seven shapes drawn in the next screen byte starting with SHAPE1. This continues until the value in BYTE is #$26, at which point the branch is not taken and the program skips to line 72 (JMP START), which starts the program from the beginning; i.e., the shape now begins its screen traversal in the first (leftmost) screen byte. Because we always follow a draw with an erase, the last shape at the right border is not left on the screen when the shape begins its new journey on the left.

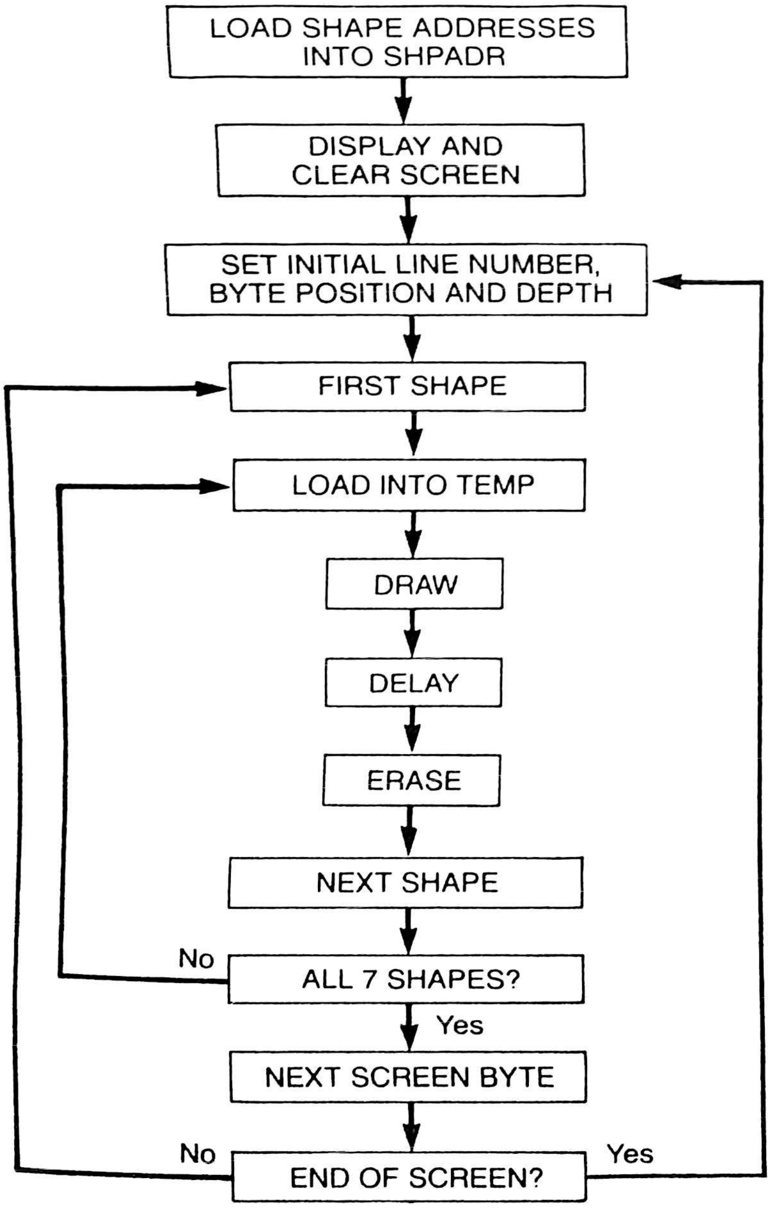
The reason for choosing #$26 for the end of screen comparison warrants some discussion, because it might seem at first glance that we should use #$27 (decimal 39) for the comparison since #$27 is in fact the last screen byte (remember screen bytes are numbered 0 to 39 or #$00 to #$27, left to right). The reason for choosing #$26 becomes apparent when we examine our shape as it approaches the right border.



Here we see the first two shapes drawn starting at screen byte #$25. It's obvious that once we get past SHAPE1, the other shapes extend into byte #$27, the last screen byte, as the shape table is 3 bytes wide. If we start at byte #$26, SHAPE2 to SHAPE7 will extend beyond byte #$27, i.e., beyond the screen border. So once the value for BYTE reaches #$26, we want to start over from the left screen position.

This discussion emphasizes the importance of examining a program in great detail before choosing numbers or instructions that seem right. The best way to debug a program is to get it right from the start, admittedly an ideal seldom realized. But if you make prior examination of details a habit, you will save yourself many headaches later on.

Finally, the DRAW routine in Program 5-1 is essentially the same as ones we've seen in previous chapters. It is a DRAW-ERASE type of routine, using EOR both to draw and erase the shape. However, instead of accessing a shape table, TEMP is accessed to obtain the shape bytes. Also, at the end of the routine, LINE is reset to its starting value (from LINEA) in preparation for the next cycle. (Remember that LINE, but not LINEA, is changed in the DRAW routine and so must be reset for the erase cycle because we want to start drawing each shape on the same line—otherwise, it wouldn't be horizontal movement.)




```
]PROGRAM 5-1
:ASM

                *1 SHAPE HORIZONTAL*
                *2 BYTES WIDE, 5 LINES DEEP
                         ORG   $6000
6000: 4C 27 60           JMP   PGM
                LINE     DS    1
                LINEA    DS    1
                BYTE     DS    1
                DEPTH    DS    1
                XCOUNT   DS    1
                SHPNO    DS    1
                DELAY    DS    1
                TEMP     DS    15
                GRAPHICS =     $C050
                MIXOFF   =     $C052
                HIRES    =     $C057
                PAGE1    =     $C054
                HIGH     =     $1B
                LOW      =     $1A
```

```
                   WAIT      =      $FCA8
                   *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                   *CONTINUE FOR ALL 7 SHAPES
6019: F7           SHPADR    DFB    #<SHAPE1
601A: 60                     DFB    #>SHAPE1
601B: 06                     DFB    #<SHAPE2
601C: 61                     DFB    #>SHAPE2
601D: 15                     DFB    #<SHAPE3
601E: 61                     DFB    #>SHAPE3
601F: 24                     DFB    #<SHAPE4
6020: 61                     DFB    #>SHAPE4
6021: 33                     DFB    #<SHAPE5
6022: 61                     DFB    #>SHAPE5
6023: 42                     DFB    #<SHAPE6
6024: 61                     DFB    #>SHAPE6
6025: 51                     DFB    #<SHAPE7
6026: 61                     DFB    #>SHAPE7
6027: AD 50 C0     PGM       LDA    GRAPHICS      ;HIRES,P.1
602A: AD 52 C0               LDA    MIXOFF
602D: AD 57 C0               LDA    HIRES
6030: AD 54 C0               LDA    PAGE1
6033: A9 00                  LDA    #$00          ;CLEAR SCREEN 1
6035: 85 1A                  STA    LOW
6037: A9 20                  LDA    #$20
6039: 85 1B                  STA    HIGH
603B: A0 00        CLR1      LDY    #$00
603D: A9 00                  LDA    #$00
603F: 91 1A        CLR       STA    (LOW),Y
6041: C8                     INY
6042: D0 FB                  BNE    CLR
6044: E6 1B                  INC    HIGH
6046: A5 1B                  LDA    HIGH
6048: C9 40                  CMP    #$40
604A: 90 EF                  BLT    CLR1
604C: A9 60                  LDA    #$60          ;LOAD DELAY
604E: 8D 09 60               STA    DELAY
                   ********** MAIN PROGRAM **********
6051: 20 7F 60     START     JSR    INITIAL       ;SET INITIAL BYTE, LINE, DEPTH
6054: A9 00        START1    LDA    #$00          ;FIRST SHAPE NUMBER
6056: 8D 08 60               STA    SHPNO
6059: 20 91 60     START2    JSR    LOADSHP       ;LOAD SHAPE INTO TEMP
605C: 20 AD 60               JSR    DRAW          ;DRAW
605F: AD 09 60               LDA    DELAY         ;DELAY
6062: 20 A8 FC               JSR    WAIT
6065: 20 AD 60               JSR    DRAW          ;ERASE
6068: EE 08 60               INC    SHPNO         ;NEXT SHAPE NUMBER
606B: AD 08 60               LDA    SHPNO
606E: C9 07                  CMP    #$07          ;FINISHED ALL 7 SHAPES?
6070: 90 E7                  BLT    START2        ;IF NO, CONTINUE WITH NEXT SHAPE
6072: EE 05 60               INC    BYTE          ;IF YES, NEXT BYTE
6075: AD 05 60               LDA    BYTE
6078: C9 26                  CMP    #$26          ;END OF SCREEN?
607A: 90 D8                  BLT    START1        ;IF NO, CONTINUE DRAW
607C: 4C 51 60               JMP    START         ;IF YES, START OVER
                   ********** SUBROUTINES **********
607F: A9 00        INITIAL   LDA    #$00
6081: 8D 05 60               STA    BYTE
6084: 8D 03 60               STA    LINE
6087: 8D 04 60               STA    LINEA
608A: 18                     CLC
608B: 69 05                  ADC    #$05          ;DEPTH OF SHAPE
```

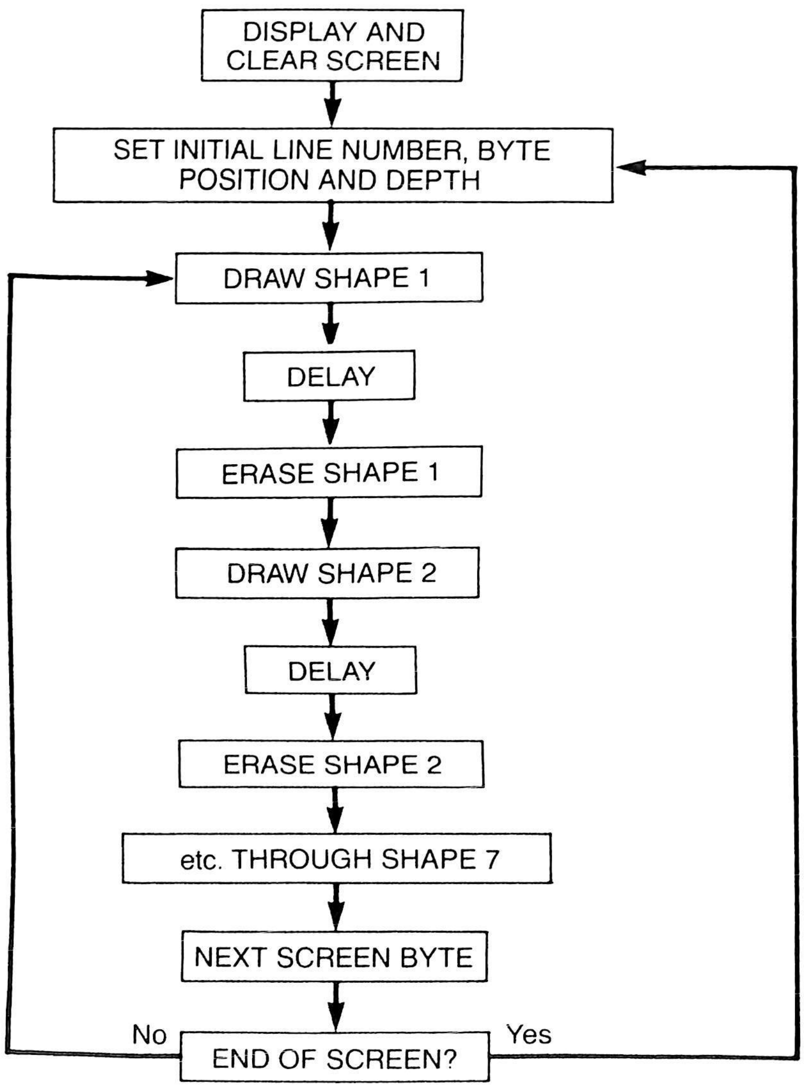
```
608D: 8D 06 60              STA  DEPTH
6090: 60                    RTS
                 **********************
6091: AD 08 60   LOADSHP    LDA  SHPNO       ;LOAD SHAPE INTO TEMP
6094: 0A                    ASL
6095: AA                    TAX
6096: BD 19 60              LDA  SHPADR,X
6099: 85 1A                 STA  LOW
609B: BD 1A 60              LDA  SHPADR+1,X
609E: 85 1B                 STA  HIGH
60A0: A0 00                 LDY  #$00
60A2: B1 1A      LOADSHP1   LDA  (LOW),Y
60A4: 99 0A 60              STA  TEMP,Y
60A7: C8                    INY
60A8: C0 0F                 CPY  #$0F
60AA: 90 F6                 BLT  LOADSHP1
60AC: 60                    RTS
                 **********************
60AD: A9 00      DRAW       LDA  #$00
60AF: 8D 07 60              STA  XCOUNT
60B2: AC 05 60   DRAW1      LDY  BYTE
60B5: AE 03 60              LDX  LINE
60B8: BD 60 61              LDA  HI,X
60BB: 85 1B                 STA  HIGH
60BD: BD 20 62              LDA  LO,X
60C0: 85 1A                 STA  LOW
60C2: AE 07 60              LDX  XCOUNT
60C5: B1 1A                 LDA  (LOW),Y
60C7: 5D 0A 60              EOR  TEMP,X
60CA: 91 1A                 STA  (LOW),Y
60CC: C8                    INY
60CD: B1 1A                 LDA  (LOW),Y
60CF: 5D 0B 60              EOR  TEMP+1,X
60D2: 91 1A                 STA  (LOW),Y
60D4: C8                    INY
60D5: B1 1A                 LDA  (LOW),Y
60D7: 5D 0C 60              EOR  TEMP+2,X
60DA: 91 1A                 STA  (LOW),Y
60DC: EE 07 60              INC  XCOUNT
60DF: EE 07 60              INC  XCOUNT
60E2: EE 07 60              INC  XCOUNT
60E5: EE 03 60              INC  LINE
60E8: AD 03 60              LDA  LINE
60EB: CD 06 60              CMP  DEPTH
60EE: 90 C2                 BLT  DRAW1
60F0: AD 04 60              LDA  LINEA
60F3: 8D 03 60              STA  LINE        ;RESET LINE FOR NEXT CYCLE
60F6: 60                    RTS
60F7: 02 00 00   SHAPE1     HEX  0200000600007E1F00 ;SHAPE TABLES
60FA: 06 00 00   7E 1F 00
6100: 7E 37 00              HEX  7E37007E7F00
6103: 7E 7F 00
6106: 04 00 00   SHAPE2     HEX  0400000C00007C3F00
6109: 0C 00 00   7C 3F 00
610F: 7C 6F 00              HEX  7C6F007C7F01
6112: 7C 7F 01
6115: 08 00 00   SHAPE3     HEX  080000180000787F00
6118: 18 00 00   78 7F 00
611E: 78 5F 01              HEX  785F01787F03
6121: 78 7F 03
6124: 10 00 00   SHAPE4     HEX  100000300000707F01
```

```
6127: 30 00 00 70 7F 01
612D: 70 3F 03   135              HEX   703F03707F07
6130: 70 7F 07
6133: 20 00 00   136   SHAPE5     HEX   200000600000607F03
6136: 60 00 00 60 7F 03
613C: 60 7F 06   137              HEX   607F06607F0F
613F: 60 7F 0F
6142: 40 00 00   138   SHAPE6     HEX   400000400100407F07
6145: 40 01 00 40 7F 07
614B: 40 7F 0D   139              HEX   407F0D407F1F
614E: 40 7F 1F
6151: 00 01 00   140   SHAPE7     HEX   000100000300007F0F
6154: 00 03 00 00 7F 0F
615A: 00 7F 1B   141              HEX   007F1B007F3F
615D: 00 7F 3F
                       HI
                       LO


736 bytes


Symbol table - numerical order:

   LOW      =$1A       HIGH     =$1B       LINE     =$6003     LINEA    =$6004
   BYTE     =$6005     DEPTH    =$6006     XCOUNT   =$6007     SHPNO    =$6008
   DELAY    =$6009     TEMP     =$600A     SHPADR   =$6019     PGM      =$6027
   CLR1     =$603B     CLR      =$603F     START    =$6051     START1   =$6054
   START2   =$6059     INITIAL  =$607F     LOADSHP  =$6091     LOADSHP1=$60A2
   DRAW     =$60AD     DRAW1    =$60B2     SHAPE1   =$60F7     SHAPE2   =$6106
   SHAPE3   =$6115     SHAPE4   =$6124     SHAPE5   =$6133     SHAPE6   =$6142
   SHAPE7   =$6151     HI       =$6160     LO       =$6220     GRAPHICS=$C050
   MIXOFF   =$C052     PAGE1    =$C054     HIRES    =$C057     WAIT     =$FCA8
```

The more astute among you might wonder why TEMP is used at all, as multiple shape tables can be accessed directly by using a counter (i.e., a number stored in the X register) with the instruction LDA SHAPE,X if the shape tables begin with a single label, SHAPE. If each of the seven shapes contains 10 bytes, the first shape can be called by LDA SHAPE,X when X = 0, the second shape when X = 10, the third shape when X = 20, etc. By manipulating X, all seven shape tables can be addressed. The problem here is that the X register (as well as the Y register and the Accumulator) can contain only a single byte, which has a maximum value of 255 decimal (#$FF hex). Thus, if the total number of bytes in all seven shape tables is large, we may have a problem if X cannot be large enough to access all the shape bytes. We would then have to use two or more counters and/or a more complicated draw routine.

Lest you think this an unusual situation, look at the shape tables in Program 5-4. Each shape contains 39 bytes; the seven shapes together contain 273 bytes, and this for shapes that are not particularly large. The use of TEMP solves this problem to a large extent. TEMP also uses a counter (see line 94 of Program 5-1), but only to load a single shape, so the limitation here is that a single shape must contain 256 bytes or less. I suppose it's conceivable that in a state of programming frenzy, you might want to draw and animate horizontally a shape that contains more than 256 bytes, although it would be so large, say 10 screen bytes wide by 30 lines deep, that it would hardly have room to move. This can be done (didn't I say at the beginning that versatility is one of the virtues of assembly language?), but not with the exact routines described here. I'll leave this to you as a problem you should be able to solve after reading this book (hint: divide the shape into less than 256 byte sections, use multiple TEMPs, and modify the draw routine).

Using TEMP and a single draw routine makes for a neat and compact program but the price we pay is an increase in program execution time because TEMP has to be loaded for each draw. Ordinarily this is not a problem, and it certainly isn't for our game program, but if extra speed is required, we can do away with TEMP and use seven different draw routines, each accessing one of the seven shape tables. This would also eliminate the need for shape address tables and counting shape numbers, and the program flow would be relatively simple—we would just draw and erase each shape in turn, testing only for the end of the screen. The program size would increase, and dramatically so with large numbers of shapes, but the program would run faster. The next program (Program 5-2) illustrates the point—it's the same as Program 5-1 but without TEMP and shape address tables.



```
]PROGRAM 5-2
:ASM
                *1 SHAPE HORIZONTAL* 7 DRAW ROUTINES
                *2 BYTES WIDE, 5 LINES DEEP
                         ORG  $6000
6000: 4C 09 60           JMP  PGM
                LINE     DS   1
                LINEA    DS   1
                BYTE     DS   1
                DEPTH    DS   1
                XCOUNT   DS   1
                DELAY    DS   1
                GRAPHICS =    $C050
                MIXOFF   =    $C052
                HIRES    =    $C057
                PAGE1    =    $C054
                HIGH     =    $1B
                LOW      =    $1A
                WAIT     =    $FCA8
6009: AD 50 C0  PGM      LDA  GRAPHICS    ;HIRES,P.1
600C: AD 52 C0           LDA  MIXOFF
600F: AD 57 C0           LDA  HIRES
6012: AD 54 C0           LDA  PAGE1
6015: A9 00              LDA  #$00        ;CLEAR SCREEN 1
6017: 85 1A              STA  LOW
6019: A9 20              LDA  #$20
601B: 85 1B              STA  HIGH
601D: A0 00     CLR1     LDY  #$00
601F: A9 00              LDA  #$00
6021: 91 1A     CLR      STA  (LOW),Y
6023: C8                 INY
6024: D0 FB              BNE  CLR
6026: E6 1B              INC  HIGH
6028: A5 1B              LDA  HIGH
602A: C9 40              CMP  #$40
602C: 90 EF              BLT  CLR1
602E: A9 60              LDA  #$60        ;LOAD DELAY
6030: 8D 08 60           STA  DELAY
                ********** MAIN PROGRAM **********
6033: 20 97 60  START    JSR  INITIAL
6036: 20 BD 60  START1   JSR  DRAW1       ;DRAW
6039: AD 08 60           LDA  DELAY
603C: 20 A8 FC           JSR  WAIT
603F: 20 BD 60           JSR  DRAW1       ;ERASE
6042: 20 F7 60           JSR  DRAW2       ;DRAW
6045: AD 08 60           LDA  DELAY
6048: 20 A8 FC           JSR  WAIT
604B: 20 F7 60           JSR  DRAW2       ;ERASE
604E: 20 31 61           JSR  DRAW3       ;DRAW
6051: AD 08 60           LDA  DELAY
6054: 20 A8 FC           JSR  WAIT
6057: 20 31 61           JSR  DRAW3       ;ERASE
605A: 20 6B 61           JSR  DRAW4       ;DRAW
605D: AD 08 60           LDA  DELAY
6060: 20 A8 FC           JSR  WAIT
6063: 20 6B 61           JSR  DRAW4       ;ERASE
6066: 20 A5 61           JSR  DRAW5       ;DRAW
6069: AD 08 60           LDA  DELAY
606C: 20 A8 FC           JSR  WAIT
606F: 20 A5 61           JSR  DRAW5       ;ERASE
6072: 20 DF 61           JSR  DRAW6       ;DRAW
```

```
6075: AD 08 60          LDA  DELAY
6078: 20 A8 FC          JSR  WAIT
607B: 20 DF 61          JSR  DRAW6        ;ERASE
607E: 20 19 62          JSR  DRAW7        ;DRAW
6081: AD 08 60          LDA  DELAY
6084: 20 A8 FC          JSR  WAIT
6087: 20 19 62          JSR  DRAW7        ;ERASE
608A: EE 05 60          INC  BYTE
608D: AD 05 60          LDA  BYTE
6090: C9 26             CMP  #$26
6092: 90 A2             BLT  START1
6094: 4C 33 60          JMP  START
                 ****** SUBROUTINES ******
6097: A9 00      INITIAL LDA  #$00
6099: 8D 05 60          STA  BYTE
609C: 8D 03 60          STA  LINE
609F: 8D 04 60          STA  LINEA
60A2: 18                CLC
60A3: 69 05             ADC  #$05         ;DEPTH OF SHAPE
60A5: 8D 06 60          STA  DEPTH
60A8: 60                RTS
                 **********************
60A9: AC 05 60   SETUP  LDY  BYTE
60AC: AE 03 60          LDX  LINE
60AF: BD BC 62          LDA  HI,X
60B2: 85 1B             STA  HIGH
60B4: BD 7C 63          LDA  LO,X
60B7: 85 1A             STA  LOW
60B9: AE 07 60          LDX  XCOUNT
60BC: 60                RTS
                 **********************
60BD: A9 00      DRAW1  LDA  #$00
60BF: 8D 07 60          STA  XCOUNT
60C2: 20 A9 60   DRAW1A JSR  SETUP
60C5: B1 1A             LDA  (LOW),Y
60C7: 5D 53 62          EOR  SHAPE1,X
60CA: 91 1A             STA  (LOW),Y
60CC: C8                INY
60CD: B1 1A             LDA  (LOW),Y
60CF: 5D 54 62          EOR  SHAPE1+1,X
60D2: 91 1A             STA  (LOW),Y
60D4: C8                INY
60D5: B1 1A             LDA  (LOW),Y
60D7: 5D 55 62          EOR  SHAPE1+2,X
60DA: 91 1A             STA  (LOW),Y
60DC: EE 07 60          INC  XCOUNT
60DF: EE 07 60          INC  XCOUNT
60E2: EE 07 60          INC  XCOUNT
60E5: EE 03 60          INC  LINE
60E8: AD 03 60          LDA  LINE
60EB: CD 06 60          CMP  DEPTH
60EE: 90 D2             BLT  DRAW1A
60F0: AD 04 60          LDA  LINEA
60F3: 8D 03 60          STA  LINE
60F6: 60                RTS
60F7: A9 00      DRAW2  LDA  #$00
60F9: 8D 07 60          STA  XCOUNT
60FC: 20 A9 60   DRAW2A JSR  SETUP
60FF: B1 1A             LDA  (LOW),Y
6101: 5D 62 62          EOR  SHAPE2,X
6104: 91 1A             STA  (LOW),Y
```

```
6106: C8                  INY
6107: B1 1A               LDA  (LOW),Y
6109: 5D 63 62            EOR  SHAPE2+1,X
610C: 91 1A               STA  (LOW),Y
610E: C8                  INY
610F: B1 1A               LDA  (LOW),Y
6111: 5D 64 62            EOR  SHAPE2+2,X
6114: 91 1A               STA  (LOW),Y
6116: EE 07 60            INC  XCOUNT
6119: EE 07 60            INC  XCOUNT
611C: EE 07 60            INC  XCOUNT
611F: EE 03 60            INC  LINE
6122: AD 03 60            LDA  LINE
6125: CD 06 60            CMP  DEPTH
6128: 90 D2               BLT  DRAW2A
612A: AD 04 60            LDA  LINEA
612D: 8D 03 60            STA  LINE
6130: 60                  RTS
6131: A9 00      DRAW3    LDA  #$00
6133: 8D 07 60            STA  XCOUNT
6136: 20 A9 60   DRAW3A   JSR  SETUP
6139: B1 1A               LDA  (LOW),Y
613B: 5D 71 62            EOR  SHAPE3,X
613E: 91 1A               STA  (LOW),Y
6140: C8                  INY
6141: B1 1A               LDA  (LOW),Y
6143: 5D 72 62            EOR  SHAPE3+1,X
6146: 91 1A               STA  (LOW),Y
6148: C8                  INY
6149: B1 1A               LDA  (LOW),Y
614B: 5D 73 62            EOR  SHAPE3+2,X
614E: 91 1A               STA  (LOW),Y
6150: EE 07 60            INC  XCOUNT
6153: EE 07 60            INC  XCOUNT
6156: EE 07 60            INC  XCOUNT
6159: EE 03 60            INC  LINE
615C: AD 03 60            LDA  LINE
615F: CD 06 60            CMP  DEPTH
6162: 90 D2               BLT  DRAW3A
6164: AD 04 60            LDA  LINEA
6167: 8D 03 60            STA  LINE
616A: 60                  RTS
616B: A9 00      DRAW4    LDA  #$00
616D: 8D 07 60            STA  XCOUNT
6170: 20 A9 60   DRAW4A   JSR  SETUP
6173: B1 1A               LDA  (LOW),Y
6175: 5D 80 62            EOR  SHAPE4,X
6178: 91 1A               STA  (LOW),Y
617A: C8                  INY
617B: B1 1A               LDA  (LOW),Y
617D: 5D 81 62            EOR  SHAPE4+1,X
6180: 91 1A               STA  (LOW),Y
6182: C8                  INY
6183: B1 1A               LDA  (LOW),Y
6185: 5D 82 62            EOR  SHAPE4+2,X
6188: 91 1A               STA  (LOW),Y
618A: EE 07 60            INC  XCOUNT
618D: EE 07 60            INC  XCOUNT
6190: EE 07 60            INC  XCOUNT
6193: EE 03 60            INC  LINE
6196: AD 03 60            LDA  LINE
```

```
6199: CD 06 60              CMP  DEPTH
619C: 90 D2                 BLT  DRAW4A
619E: AD 04 60              LDA  LINEA
61A1: 8D 03 60              STA  LINE
61A4: 60                    RTS
61A5: A9 00        DRAW5    LDA  #$00
61A7: 8D 07 60              STA  XCOUNT
61AA: 20 A9 60     DRAW5A   JSR  SETUP
61AD: B1 1A                 LDA  (LOW),Y
61AF: 5D 8F 62              EOR  SHAPE5,X
61B2: 91 1A                 STA  (LOW),Y
61B4: C8                    INY
61B5: B1 1A                 LDA  (LOW),Y
61B7: 5D 90 62              EOR  SHAPE5+1,X
61BA: 91 1A                 STA  (LOW),Y
61BC: C8                    INY
61BD: B1 1A                 LDA  (LOW),Y
61BF: 5D 91 62              EOR  SHAPE5+2,X
61C2: 91 1A                 STA  (LOW),Y
61C4: EE 07 60              INC  XCOUNT
61C7: EE 07 60              INC  XCOUNT
61CA: EE 07 60              INC  XCOUNT
61CD: EE 03 60              INC  LINE
61D0: AD 03 60              LDA  LINE
61D3: CD 06 60              CMP  DEPTH
61D6: 90 D2                 BLT  DRAW5A
61D8: AD 04 60              LDA  LINEA
61DB: 8D 03 60              STA  LINE
61DE: 60                    RTS
61DF: A9 00        DRAW6    LDA  #$00
61E1: 8D 07 60              STA  XCOUNT
61E4: 20 A9 60     DRAW6A   JSR  SETUP
61E7: B1 1A                 LDA  (LOW),Y
61E9: 5D 9E 62              EOR  SHAPE6,X
61EC: 91 1A                 STA  (LOW),Y
61EE: C8                    INY
61EF: B1 1A                 LDA  (LOW),Y
61F1: 5D 9F 62              EOR  SHAPE6+1,X
61F4: 91 1A                 STA  (LOW),Y
61F6: C8                    INY
61F7: B1 1A                 LDA  (LOW),Y
61F9: 5D A0 62              EOR  SHAPE6+2,X
61FC: 91 1A                 STA  (LOW),Y
61FE: EE 07 60              INC  XCOUNT
6201: EE 07 60              INC  XCOUNT
6204: EE 07 60              INC  XCOUNT
6207: EE 03 60              INC  LINE
620A: AD 03 60              LDA  LINE
620D: CD 06 60              CMP  DEPTH
6210: 90 D2                 BLT  DRAW6A
6212: AD 04 60              LDA  LINEA
6215: 8D 03 60              STA  LINE
6218: 60                    RTS
6219: A9 00        DRAW7    LDA  #$00
621B: 8D 07 60              STA  XCOUNT
621E: 20 A9 60     DRAW7A   JSR  SETUP
6221: B1 1A                 LDA  (LOW),Y
6223: 5D AD 62              EOR  SHAPE7,X
6226: 91 1A                 STA  (LOW),Y
6228: C8                    INY
6229: B1 1A                 LDA  (LOW),Y
```

```
622B: 5D AE 62              EOR  SHAPE7+1,X
622E: 91 1A                 STA  (LOW),Y
6230: C8                    INY
6231: B1 1A                 LDA  (LOW),Y
6233: 5D AF 62              EOR  SHAPE7+2,X
6236: 91 1A                 STA  (LOW),Y
6238: EE 07 60              INC  XCOUNT
623B: EE 07 60              INC  XCOUNT
623E: EE 07 60              INC  XCOUNT
6241: EE 03 60              INC  LINE
6244: AD 03 60              LDA  LINE
6247: CD 06 60              CMP  DEPTH
624A: 90 D2                 BLT  DRAW7A
624C: AD 04 60              LDA  LINEA
624F: 8D 03 60              STA  LINE
6252: 60                    RTS
6253: 02 00 00      SHAPE1  HEX  0200000600007E1F00 ;SHAPE TABLES
6256: 06 00 00 7E 1F 00
625C: 7E 37 00              HEX  7E37007E7F00
625F: 7E 7F 00
6262: 04 00 00      SHAPE2  HEX  0400000C00007C3F00
6265: 0C 00 00 7C 3F 00
626B: 7C 6F 00              HEX  7C6F007C7F01
626E: 7C 7F 01
6271: 08 00 00      SHAPE3  HEX  080000180000787F00
6274: 18 00 00 78 7F 00
627A: 78 5F 01              HEX  785F01787F03
627D: 78 7F 03
6280: 10 00 00      SHAPE4  HEX  100000300000707F01
6283: 30 00 00 70 7F 01
6289: 70 3F 03              HEX  703F03707F07
628C: 70 7F 07
628F: 20 00 00      SHAPE5  HEX  200000600000607F03
6292: 60 00 00 60 7F 03
6298: 60 7F 06              HEX  607F06607F0F
629B: 60 7F 0F
629E: 40 00 00      SHAPE6  HEX  400000400100407F07
62A1: 40 01 00 40 7F 07
62A7: 40 7F 0D              HEX  407F0D407F1F
62AA: 40 7F 1F
62AD: 00 01 00      SHAPE7  HEX  000100000300007F0F
62B0: 00 03 00 00 7F 0F
62B6: 00 7F 1B              HEX  007F1B007F3F
62B9: 00 7F 3F
                    HI
                    LO

1084 bytes
```

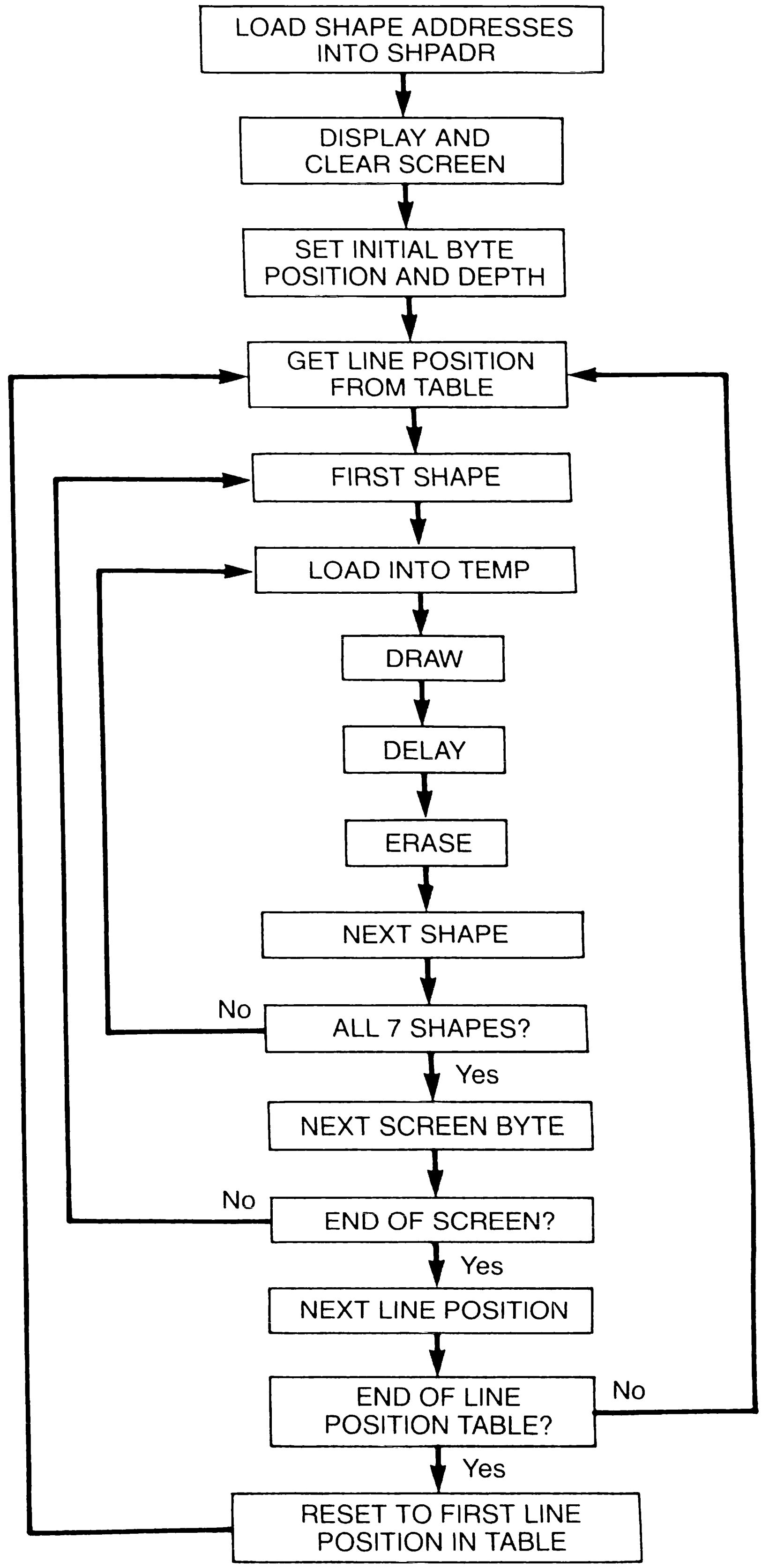
```
Symbol table - numerical order:

     LOW      =$1A       HIGH     =$1B       LINE     =$6003       LINEA    =$6004
     BYTE     =$6005     DEPTH    =$6006     XCOUNT   =$6007       DELAY    =$6008
     PGM      =$6009     CLR1     =$601D     CLR      =$6021       START    =$6033
     START1   =$6036     INITIAL  =$6097     SETUP    =$60A9       DRAW1    =$60BD
     DRAW1A   =$60C2     DRAW2    =$60F7     DRAW2A   =$60FC       DRAW3    =$6131
     DRAW3A   =$6136     DRAW4    =$616B     DRAW4A   =$6170       DRAW5    =$61A5
     DRAW5A   =$61AA     DRAW6    =$61DF     DRAW6A   =$61E4       DRAW7    =$6219
     DRAW7A   =$621E     SHAPE1   =$6253     SHAPE2   =$6262       SHAPE3   =$6271
     SHAPE4   =$6280     SHAPE5   =$628F     SHAPE6   =$629E       SHAPE7   =$62AD
     HI       =$62BC     LO       =$637C     GRAPHICS=$C050        MIXOFF   =$C052
     PAGE1    =$C054     HIRES    =$C057     WAIT     =$FCA8
```

As you can see, Program 5-2 is larger than Program 5-1. With more shapes, it would be larger still, but it does run faster, even though it doesn't seem to—the plane traverses the screen in about the same time for both programs but this is because the programs are simple, with only one shape, and so the determining factor is the time delay. The speed difference would be noticeable only with larger and more complicated programs.

SHAPES AT NEW LINE POSITIONS

Before going on to the next section, allow a minor digression—I want to illustrate how easy it is to modify our programs to make them more interesting. In the next program (5-3) we're going to modify Program 5-1 so that the airplanes begin their screen traversal at five different line positions instead of at the same line each time. This makes for a more visually appealing program.

The modifications are fairly simple. In the INITIAL subroutine, the starting line for each traversal is determined by accessing numbers in a table labelled NEWLINE that contains five bytes, one for each new line position. The bytes are selected by LDA NEWLINE,X (line 86) where X contains values 0 to 4. The values for X are loaded from a reserved memory location labeled COUNTER (LDX COUNTER, line 85). Initially, COUNTER is set to 0 (lines 79 and 80) and is incremented by one at the end of each screen traversal (line 75). When the values in COUNTER are from 0 to 4, the branch in line 78 is taken and the shape starts at a new line position. When the value in COUNTER reaches 5, we're at the end of the NEWLINE table and so we want to start over. At this point, the branch at line 78 is not taken and COUNTER is reset to 0 before we begin the next screen journey (lines 79 to 81). To program more or less line positions, place the desired values in NEWLINE and change the CMP value in line 77 to the number of bytes in the NEWLINE table.


LOAD SHAPE ADDRESSES INTO SHPADR
DISPLAY AND CLEAR SCREEN
SET INITIAL BYTE POSITION AND DEPTH
GET LINE POSITION FROM TABLE
FIRST SHAPE
LOAD INTO TEMP
DRAW
DELAY
ERASE
NEXT SHAPE
ALL 7 SHAPES?
No
Yes
NEXT SCREEN BYTE
END OF SCREEN?
No
Yes
NEXT LINE POSITION
END OF LINE POSITION TABLE?
No
Yes
RESET TO FIRST LINE POSITION IN TABLE

```
]PROGRAM 5-3
:ASM
                  *1 SHAPE HORIZONTAL NEWLINE*
                  *2 BYTES WIDE, 5 LINES DEEP
                           ORG  $6000
6000: 4C 28 60             JMP  PGM
                  LINE     DS   1
                  LINEA    DS   1
                  BYTE     DS   1
                  DEPTH    DS   1
                  XCOUNT   DS   1
                  SHPNO    DS   1
                  DELAY    DS   1
                  COUNTER  DS   1
                  TEMP     DS   15
                  GRAPHICS =    $C050
                  MIXOFF   =    $C052
                  HIRES    =    $C057
                  PAGE1    =    $C054
                  HIGH     =    $1B
                  LOW      =    $1A
                  WAIT     =    $FCA8
                  *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
601A: 17          *CONTINUE FOR ALL 7 SHAPES
601B: 61          SHPADR   DFB  #<SHAPE1
601C: 26                   DFB  #>SHAPE1
601D: 61                   DFB  #<SHAPE2
601E: 35                   DFB  #>SHAPE2
601F: 61                   DFB  #<SHAPE3
6020: 44                   DFB  #>SHAPE3
6021: 61                   DFB  #<SHAPE4
6022: 53                   DFB  #>SHAPE4
6023: 61                   DFB  #<SHAPE5
6024: 62                   DFB  #>SHAPE5
6025: 61                   DFB  #<SHAPE6
6026: 71                   DFB  #>SHAPE6
6027: 61                   DFB  #<SHAPE7
6028: AD 50 C0             DFB  #>SHAPE7
602B: AD 52 C0    PGM      LDA  GRAPHICS     ;HIRES,P.1
602E: AD 57 C0             LDA  MIXOFF
6031: AD 54 C0             LDA  HIRES
6034: A9 00                LDA  PAGE1
6036: 85 1A                LDA  #$00         ;CLEAR SCREEN 1
6038: A9 20                STA  LOW
603A: 85 1B                LDA  #$20
603C: A0 00                STA  HIGH
603E: A9 00       CLR1     LDY  #$00
6040: 91 1A                LDA  #$00
6042: C8          CLR      STA  (LOW),Y
6043: D0 FB                INY
6045: E6 1B                BNE  CLR
6047: A5 1B                INC  HIGH
6049: C9 40                LDA  HIGH
604B: 90 EF                CMP  #$40
604D: A9 60                BLT  CLR1
604F: 8D 09 60             LDA  #$60         ;LOAD DELAY
6052: A9 00                STA  DELAY
6054: 8D 0A 60             LDA  #$00         ;ZERO COUNTER
                           STA  COUNTER
                  ********** MAIN PROGRAM **********
6057: 20 94 60    START    JSR  INITIAL      ;SET INITIAL BYTE, LINE, DEPTH
```

```
605A: A9 00        START1   LDA  #$00        ;FIRST SHAPE NUMBER
605C: 8D 08 60              STA  SHPNO
605F: 20 AC 60     START2   JSR  LOADSHP     ;LOAD SHAPE INTO TEMP
6062: 20 C8 60              JSR  DRAW        ;DRAW
6065: AD 09 60              LDA  DELAY       ;DELAY
6068: 20 A8 FC              JSR  WAIT
606B: 20 C8 60              JSR  DRAW        ;ERASE
606E: EE 08 60              INC  SHPNO       ;NEXT SHAPE NUMBER
6071: AD 08 60              LDA  SHPNO
6074: C9 07                 CMP  #$07        ;FINISHED ALL 7 SHAPES?
6076: 90 E7                 BLT  START2      ;IF NO, CONTINUE WITH NEXT SHAPE
6078: EE 05 60              INC  BYTE        ;IF YES, NEXT BYTE
607B: AD 05 60              LDA  BYTE
607E: C9 26                 CMP  #$26        ;END OF SCREEN?
6080: 90 D8                 BLT  START1      ;IF NO, CONTINUE DRAW
6082: EE 0A 60              INC  COUNTER     ;IF YES, INCREMENT COUNTER
6085: AD 0A 60              LDA  COUNTER
6088: C9 05                 CMP  #$05        ;FINISHED 5 LINES?
608A: 90 05                 BLT  CONT        ;IF NO, CONTINUE
608C: A9 00                 LDA  #$00        ;IF YES, RESET COUNTER
608E: 8D 0A 60              STA  COUNTER         TO ZERO AND
6091: 4C 57 60     CONT     JMP  START            START OVER
                   ********** SUBROUTINES **********
6094: A9 00        INITIAL  LDA  #$00
6096: 8D 05 60              STA  BYTE
6099: AE 0A 60              LDX  COUNTER
609C: BD 12 61              LDA  NEWLINE,X
609F: 8D 03 60              STA  LINE
60A2: 8D 04 60              STA  LINEA
60A5: 18                    CLC
60A6: 69 05                 ADC  #$05        ;DEPTH OF SHAPE
60A8: 8D 06 60              STA  DEPTH
60AB: 60                    RTS
                   **********************
60AC: AD 08 60     LOADSHP  LDA  SHPNO       ;LOAD SHAPE INTO TEMP
60AF: 0A                    ASL
60B0: AA                    TAX
60B1: BD 1A 60              LDA  SHPADR,X
60B4: 85 1A                 STA  LOW
60B6: BD 1B 60              LDA  SHPADR+1,X
60B9: 85 1B                 STA  HIGH
60BB: A0 00                 LDY  #$00
60BD: B1 1A        LOADSHP1 LDA  (LOW),Y
60BF: 99 0B 60              STA  TEMP,Y
60C2: C8                    INY
60C3: C0 0F                 CPY  #$0F
60C5: 90 F6                 BLT  LOADSHP1
60C7: 60                    RTS
                   ***********************
60C8: A9 00        DRAW     LDA  #$00
60CA: 8D 07 60              STA  XCOUNT
60CD: AC 05 60     DRAW1    LDY  BYTE
60D0: AE 03 60              LDX  LINE
60D3: BD 80 61              LDA  HI,X
60D6: 85 1B                 STA  HIGH
60D8: BD 40 62              LDA  LO,X
60DB: 85 1A                 STA  LOW
60DD: AE 07 60              LDX  XCOUNT
60E0: B1 1A                 LDA  (LOW),Y
60E2: 5D 0B 60              EOR  TEMP,X
60E5: 91 1A                 STA  (LOW),Y
```

```
60E7: C8                        INY
60E8: B1 1A                     LDA   (LOW),Y
60EA: 5D 0C 60                  EOR   TEMP+1,X
60ED: 91 1A                     STA   (LOW),Y
60EF: C8                        INY
60F0: B1 1A                     LDA   (LOW),Y
60F2: 5D 0D 60                  EOR   TEMP+2,X
60F5: 91 1A                     STA   (LOW),Y
60F7: EE 07 60                  INC   XCOUNT
60FA: EE 07 60                  INC   XCOUNT
60FD: EE 07 60                  INC   XCOUNT
6100: EE 03 60                  INC   LINE
6103: AD 03 60                  LDA   LINE
6106: CD 06 60                  CMP   DEPTH
6109: 90 C2                     BLT   DRAW1
610B: AD 04 60                  LDA   LINEA
610E: 8D 03 60                  STA   LINE        ;RESET LINE FOR NEXT CYCLE
6111: 60                        RTS
6112: 00 A0 14        NEWLINE   HEX   00A0143060
6115: 30 60
6117: 02 00 00        SHAPE1    HEX   0200000600007E1F00 ;SHAPE TABLES
611A: 06 00 00 7E 1F 00
6120: 7E 37 00                  HEX   7E37007E7F00
6123: 7E 7F 00
6126: 04 00 00        SHAPE2    HEX   0400000C00007C3F00
6129: 0C 00 00 7C 3F 00
612F: 7C 6F 00                  HEX   7C6F007C7F01
6132: 7C 7F 01
6135: 08 00 00        SHAPE3    HEX   080000180000787F00
6138: 18 00 00 78 7F 00
613E: 78 5F 01                  HEX   785F01787F03
6141: 78 7F 03
6144: 10 00 00        SHAPE4    HEX   100000300000707F01
6147: 30 00 00 70 7F 01
614D: 70 3F 03                  HEX   703F03707F07
6150: 70 7F 07
6153: 20 00 00        SHAPE5    HEX   200000600000607F03
6156: 60 00 00 60 7F 03
615C: 60 7F 06                  HEX   607F06607F0F
615F: 60 7F 0F
6162: 40 00 00        SHAPE6    HEX   400000400100407F07
6165: 40 01 00 40 7F 07
616B: 40 7F 0D                  HEX   407F0D407F1F
616E: 40 7F 1F
6171: 00 01 00        SHAPE7    HEX   000100000300007F0F
6174: 00 03 00 00 7F 0F
617A: 00 7F 1B                  HEX   007F1B007F3F
617D: 00 7F 3F
```

HI
LO

768 bytes

Symbol table - numerical order:

```
LOW      =$1A      HIGH     =$1B      LINE     =$6003    LINEA    =$6004
BYTE     =$6005    DEPTH    =$6006    XCOUNT   =$6007    SHPNO    =$6008
DELAY    =$6009    COUNTER  =$600A    TEMP     =$600B    SHPADR   =$601A
PGM      =$6028    CLR1     =$603C    CLR      =$6040    START    =$6057
START1   =$605A    START2   =$605F    CONT     =$6091    INITIAL  =$6094
LOADSHP  =$60AC    LOADSHP1=$60BD     DRAW     =$60C8    DRAW1    =$60CD
NEWLINE  =$6112    SHAPE1   =$6117    SHAPE2   =$6126    SHAPE3   =$6135
SHAPE4   =$6144    SHAPE5   =$6153    SHAPE6   =$6162    SHAPE7   =$6171
HI       =$6180    LO       =$6240    GRAPHICS=$C050     MIXOFF   =$C052
PAGE1    =$C054    HIRES    =$C057    WAIT     =$FCA8
```

DRAW-DRAW

As a special added attraction, for your edification and programming pleasure, I hereby present Program 5-4, which is the same as Program 5-1, except it uses a DRAW-DRAW routine instead of DRAW-ERASE. Let's look at some of the differences between Program 5-1 and 5-4.

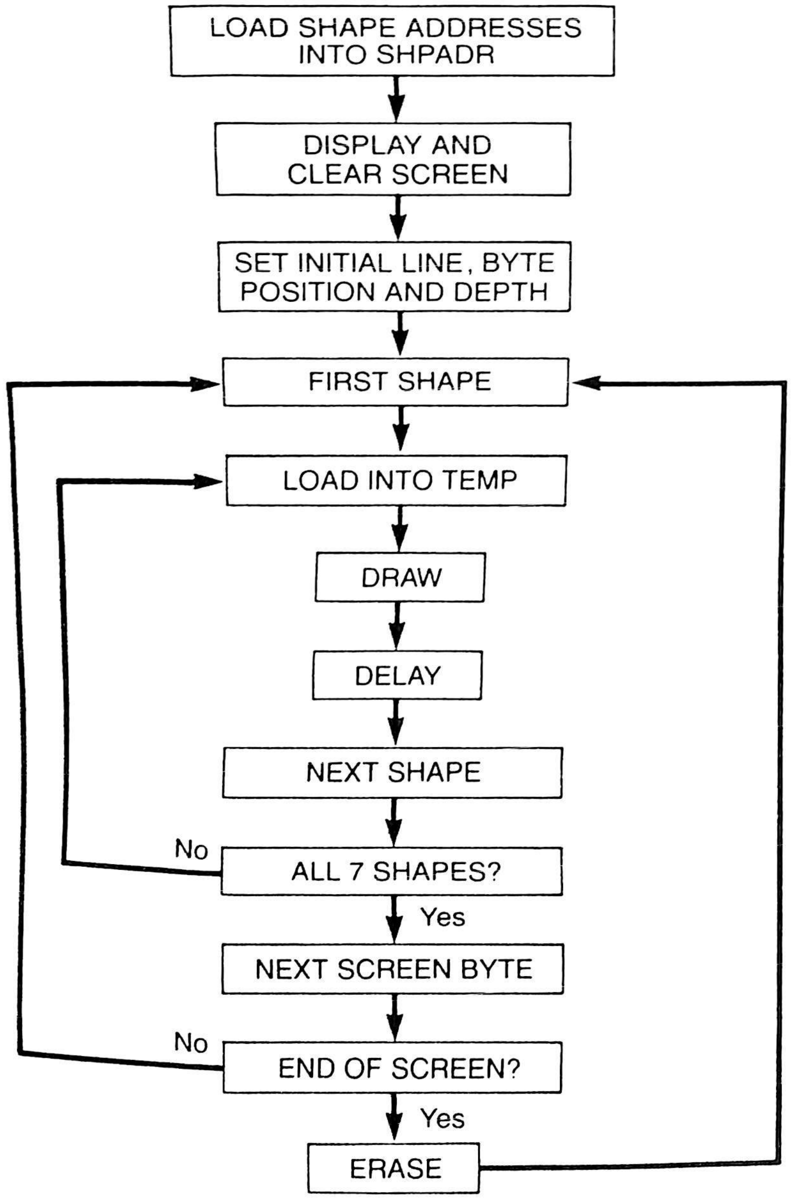
First, since there is no erase step, attention has to be paid to the shape tables to insure that no part of a shape is left on the screen. For vertical animation, we include a border of #$00's equal to the maximum shape move. For horizontal animation, the situation is somewhat different. If we look at the shape tables at the beginning of this chapter, we see that the last shape (number 6) has no bits in the first byte. Thus when we continue with shape 0 in the second byte, shape 6 is completely erased. Fine. But suppose we drew the shape starting with the first column instead of the second. Shape 6 would then have bits in the first byte. If we then continue with shape 0 in the second byte, the bits in the first byte would remain on the screen. Solution? Draw the shapes so the first byte is empty at shape 6—otherwise, a trailing whole byte of #$00's will have to be included. This not only would increase the size of the shape tables but would also mean that a shape could not start at a screen border but rather one byte over.

Next, the DRAW routine does not use EOR instructions, but rather plots by LDA shape byte, STA screen location. Note also that the shape bytes are retrieved from TEMP rather than from the shape tables directly, as in Program 5-1. Because we want to erase the shape when it reaches the right border in preparation for a new screen traversal, we do need a separate ERASE routine. This ERASE routine is essentially identical to the DRAW routine of Program 5-1; i.e., it erases using the EOR method of plotting because when accessed, the screen bytes already contain the shape bytes to be erased.

Finally, let's examine some details in the MAIN PROGRAM. First we initialize the shape position and depth, select the first shape, load TEMP, draw, and then delay. We do not erase as was done in Program 5-1, as the erase is necessary only when the shape has reached the screen border. We continue by testing to see if we've drawn all seven shapes and, if we have, we start again at the next screen byte; this continues until we've reached the end of the screen (BYTE = #$26). At this point, before going to START to begin a new screen traversal, we go to the ERASE routine to erase the last shape. Note that the last shape is drawn starting in screen byte #$25 but BYTE contains the value #$26. So the first line in the ERASE routine is DEC BYTE, which puts #$25 in BYTE in preparation for the erase. The last line of ERASE then sends the program to START for a new beginning.

If we run Programs 5-1 and 5-4, we see very little difference with perhaps somewhat smoother animation in 5-4 on close inspection. The price we pay for this is a somewhat longer program because of the extra erase routine. Again, the choice of DRAW-ERASE or DRAW-DRAW depends on the program's particular requirements. With a larger, more complicated shape, the smoothness inherent in DRAW-DRAW may become more apparent and, of course, if the program doesn't remove shapes from the screen, the extra erase routine would not be needed. On the other hand, Program 5-4 would not be appropriate if the shape were involved in collision detection or were to be drawn over a background.

Programs 5-2, 5-3, and 5-4 are not incorporated into the final game program because I would like you to use them as starting points to ease you into attempting your own modifications to the game once you've finished Part One. Suggestions for modifications and the problems to consider will be discussed in the last chapter.




```
]PROGRAM 5-4
:ASM

                 *1 SHAPE HORIZONTAL - DRAW-DRAW*
                 *2 BYTES WIDE, 5 LINES DEEP
                          ORG    $6000
6000: 4C 27 60            JMP    PGM
                 LINE     DS     1
                 LINEA    DS     1
                 BYTE     DS     1
                 DEPTH    DS     1
                 XCOUNT   DS     1
                 SHPNO    DS     1
                 DELAY    DS     1
                 TEMP     DS     15
                 GRAPHICS =      $C050
                 MIXOFF   =      $C052
                 HIRES    =      $C057
                 PAGE1    =      $C054
                 HIGH     =      $1B
                 LOW      =      $1A
```

```
                WAIT      =    $FCA8
                *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                *CONTINUE FOR ALL 7 SHAPES
6019: 3D        SHPADR  DFB  #<SHAPE1
601A: 61                DFB  #>SHAPE1
601B: 4C                DFB  #<SHAPE2
601C: 61                DFB  #>SHAPE2
601D: 5B                DFB  #<SHAPE3
601E: 61                DFB  #>SHAPE3
601F: 6A                DFB  #<SHAPE4
6020: 61                DFB  #>SHAPE4
6021: 79                DFB  #<SHAPE5
6022: 61                DFB  #>SHAPE5
6023: 88                DFB  #<SHAPE6
6024: 61                DFB  #>SHAPE6
6025: 97                DFB  #<SHAPE7
6026: 61                DFB  #>SHAPE7
6027: AD 50 C0  PGM     LDA  GRAPHICS    ;HIRES,P.1
602A: AD 52 C0          LDA  MIXOFF
602D: AD 57 C0          LDA  HIRES
6030: AD 54 C0          LDA  PAGE1
6033: A9 00             LDA  #$00        ;CLEAR SCREEN 1
6035: 85 1A             STA  LOW
6037: A9 20             LDA  #$20
6039: 85 1B             STA  HIGH
603B: A0 00     CLR1    LDY  #$00
603D: A9 00             LDA  #$00
603F: 91 1A     CLR     STA  (LOW),Y
6041: C8                INY
6042: D0 FB             BNE  CLR
6044: E6 1B             INC  HIGH
6046: A5 1B             LDA  HIGH
6048: C9 40             CMP  #$40
604A: 90 EF             BLT  CLR1
604C: A9 60             LDA  #$60        ;LOAD DELAY
604E: 8D 09 60          STA  DELAY
                ********** MAIN PROGRAM **********
6051: 20 7C 60  START   JSR  INITIAL     ;SET INITIAL BYTE, LINE, DEPTH
6054: A9 00     START1  LDA  #$00        ;FIRST SHAPE NUMBER
6056: 8D 08 60          STA  SHPNO
6059: 20 8E 60  START2  JSR  LOADSHP     ;LOAD SHAPE INTO TEMP
605C: 20 AA 60          JSR  DRAW        ;DRAW
605F: AD 09 60          LDA  DELAY       ;DELAY
6062: 20 A8 FC          JSR  WAIT
6065: EE 08 60          INC  SHPNO       ;NEXT SHAPE NUMBER
6068: AD 08 60          LDA  SHPNO
606B: C9 07             CMP  #$07        ;FINISHED ALL 7 SHAPES?
606D: 90 EA             BLT  START2      ;IF NO, CONTINUE WITH NEXT SHAPE
606F: EE 05 60          INC  BYTE        ;IF YES, NEXT BYTE
6072: AD 05 60          LDA  BYTE
6075: C9 26             CMP  #$26        ;END OF SCREEN?
6077: 90 DB             BLT  START1      ;IF NO, CONTINUE DRAW
6079: 4C EE 60          JMP  ERASE       ;IF YES, ERASE & START OVER
                ********** SUBROUTINES **********
607C: A9 00     INITIAL LDA  #$00
607E: 8D 05 60          STA  BYTE
6081: 8D 03 60          STA  LINE
6084: 8D 04 60          STA  LINEA
6087: 18                CLC
6088: 69 05             ADC  #$05        ;DEPTH OF SHAPE
608A: 8D 06 60          STA  DEPTH
```

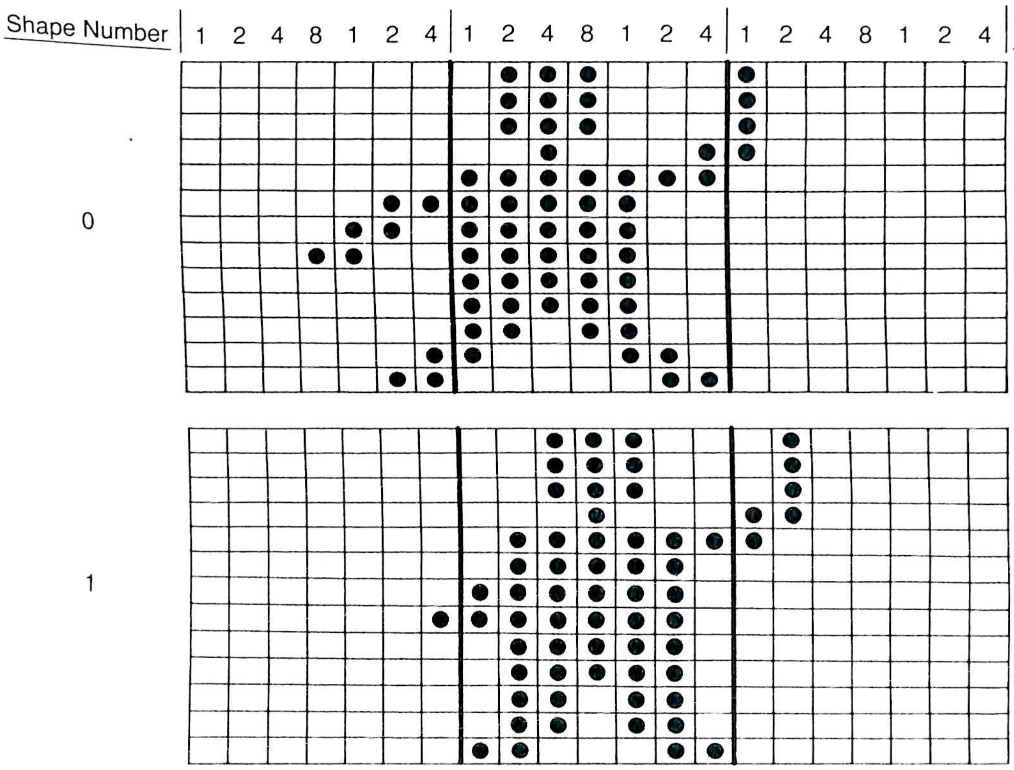
```
608D: 60                       RTS
                    **********************
608E: AD 08 60      LOADSHP    LDA  SHPNO        ;LOAD SHAPE INTO TEMP
6091: 0A                       ASL
6092: AA                       TAX
6093: BD 19 60                 LDA  SHPADR,X
6096: 85 1A                    STA  LOW
6098: BD 1A 60                 LDA  SHPADR+1,X
609B: 85 1B                    STA  HIGH
609D: A0 00                    LDY  #$00
609F: B1 1A         LOADSHP1   LDA  (LOW),Y
60A1: 99 0A 60                 STA  TEMP,Y
60A4: C8                       INY
60A5: C0 0F                    CPY  #$0F
60A7: 90 F6                    BLT  LOADSHP1
60A9: 60                       RTS
                    **********************
60AA: A9 00         DRAW       LDA  #$00
60AC: 8D 07 60                 STA  XCOUNT
60AF: AC 05 60      DRAW1      LDY  BYTE
60B2: AE 03 60                 LDX  LINE
60B5: BD A6 61                 LDA  HI,X
60B8: 85 1B                    STA  HIGH
60BA: BD 66 62                 LDA  LO,X
60BD: 85 1A                    STA  LOW
60BF: AE 07 60                 LDX  XCOUNT
60C2: BD 0A 60                 LDA  TEMP,X
60C5: 91 1A                    STA  (LOW),Y
60C7: C8                       INY
60C8: BD 0B 60                 LDA  TEMP+1,X
60CB: 91 1A                    STA  (LOW),Y
60CD: C8                       INY
60CE: BD 0C 60                 LDA  TEMP+2,X
60D1: 91 1A                    STA  (LOW),Y
60D3: EE 07 60                 INC  XCOUNT
60D6: EE 07 60                 INC  XCOUNT
60D9: EE 07 60                 INC  XCOUNT
60DC: EE 03 60                 INC  LINE
60DF: AD 03 60                 LDA  LINE
60E2: CD 06 60                 CMP  DEPTH
60E5: 90 C8                    BLT  DRAW1
60E7: AD 04 60                 LDA  LINEA
60EA: 8D 03 60                 STA  LINE         ;RESET LINE FOR NEXT CYCLE
60ED: 60                       RTS
                    *************************
60EE: CE 05 60      ERASE      DEC  BYTE
60F1: A9 00                    LDA  #$00
60F3: 8D 07 60                 STA  XCOUNT
60F6: AC 05 60      ERASE1     LDY  BYTE
60F9: AE 03 60                 LDX  LINE
60FC: BD A6 61                 LDA  HI,X
60FF: 85 1B                    STA  HIGH
6101: BD 66 62                 LDA  LO,X
6104: 85 1A                    STA  LOW
6106: AE 07 60                 LDX  XCOUNT
6109: B1 1A                    LDA  (LOW),Y
610B: 5D 0A 60                 EOR  TEMP,X
610E: 91 1A                    STA  (LOW),Y
6110: C8                       INY
6111: B1 1A                    LDA  (LOW),Y
6113: 5D 0B 60                 EOR  TEMP+1,X
```

```
6116: 91 1A      141          STA  (LOW),Y
6118: C8         142          INY
6119: B1 1A      143          LDA  (LOW),Y
611B: 5D 0C 60   144          EOR  TEMP+2,X
611E: 91 1A      145          STA  (LOW),Y
6120: EE 07 60   146          INC  XCOUNT
6123: EE 07 60   147          INC  XCOUNT
6126: EE 07 60   148          INC  XCOUNT
6129: EE 03 60   149          INC  LINE
612C: AD 03 60   150          LDA  LINE
612F: CD 06 60   151          CMP  DEPTH
6132: 90 C2      152          BLT  ERASE1
6134: AD 04 60   153          LDA  LINEA
6137: 8D 03 60   154          STA  LINE
613A: 4C 51 60   155          JMP  START
613D: 02 00 00   156  SHAPE1  HEX  0200000600007E1F00 ;SHAPE TABLES
6140: 06 00 00 7E 1F 00
6146: 7E 37 00   157          HEX  7E37007E7F00
6149: 7E 7F 00
614C: 04 00 00   158  SHAPE2  HEX  0400000C00007C3F00
614F: 0C 00 00 7C 3F 00
6155: 7C 6F 00   159          HEX  7C6F007C7F01
6158: 7C 7F 01
615B: 08 00 00   160  SHAPE3  HEX  080000180000787F00
615E: 18 00 00 78 7F 00
6164: 78 5F 01   161          HEX  785F01787F03
6167: 78 7F 03
616A: 10 00 00   162  SHAPE4  HEX  100000300000707F01
616D: 30 00 00 70 7F 01
6173: 70 3F 03   163          HEX  703F03707F07
6176: 70 7F 07
6179: 20 00 00   164  SHAPE5  HEX  200000600000607F03
617C: 60 00 00 60 7F 03
6182: 60 7F 06   165          HEX  607F06607F0F
6185: 60 7F 0F
6188: 40 00 00   166  SHAPE6  HEX  400000400100407F07
618B: 40 01 00 40 7F 07
6191: 40 7F 0D   167          HEX  407F0D407F1F
6194: 40 7F 1F
6197: 00 01 00   168  SHAPE7  HEX  000100000300007F0F
619A: 00 03 00 00 7F 0F
61A0: 00 7F 1B   169          HEX  007F1B007F3F
61A3: 00 7F 3F
                      HI
                      LO

806 bytes

Symbol table - numerical order:

  LOW     =$1A      HIGH     =$1B      LINE     =$6003   LINEA    =$6004
  BYTE    =$6005    DEPTH    =$6006    XCOUNT   =$6007   SHPNO    =$6008
  DELAY   =$6009    TEMP     =$600A    SHPADR   =$6019   PGM      =$6027
  CLR1    =$603B    CLR      =$603F    START    =$6051   START1   =$6054
  START2  =$6059    INITIAL  =$607C    LOADSHP  =$608E   LOADSHP1=$609F
  DRAW    =$60AA    DRAW1    =$60AF    ERASE    =$60EE   ERASE1   =$60F6
  SHAPE1  =$613D    SHAPE2   =$614C    SHAPE3   =$615B   SHAPE4   =$616A
  SHAPE5  =$6179    SHAPE6   =$6188    SHAPE7   =$6197   HI       =$61A6
  LO      =$6266    GRAPHICS=$C050     MIXOFF   =$C052   PAGE1    =$C054
  HIRES   =$C057    WAIT     =$FCA8
```

INTERNAL ANIMATION

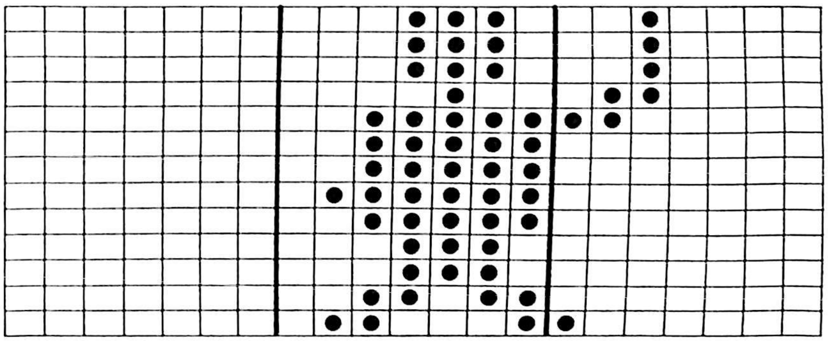
Internal animation refers to movement of parts of a shape as the shape itself moves (or doesn't move) around the screen. For example, if we're moving a person shape around, we might want to move his (her) arms and legs to simulate walking or running. This is exactly what we're going to do in the next program (5-5).

The trick to internal animation is simply to have different shape tables displaying various parts of the shape in different positions. This can be done with any type of general movement—vertical, horizontal, diagonal, or curved—or even if the shape is standing still, but it is applied most naturally to horizontal movement, because such movement requires different shape tables anyway. Program 5-5 is virtually identical to Program 5-1 except that the shape is now a person and the seven shape tables display arms and legs in different positions. When these shapes are displayed sequentially, the illusion of walking is produced. The only other change is that a line is drawn along the bottom of the screen (see lines 55 to 65) at screen line #$B7 (decimal 183) so that the person has something to walk on. You could omit the line and have the person walk on air (with a smile on his/her face?), but both the line and the shape tables are going to be incorporated into the final game program, so let's leave it the way it is. Here are the seven shapes for Program 5-5. (One minor note: one arm is shown pointing up and not moving—this is the arm that carries the gun with which the person is going to shoot at airplanes—who said game designs have to make sense?)



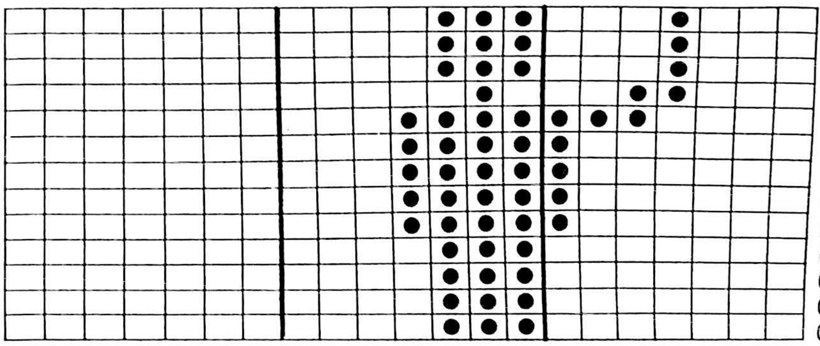
Shape Number	Shape Tables		
0	00	0E	01
	00	0E	01
	00	0E	01
	00	44	01
	00	7F	00
	60	1F	00
	30	1F	00
	18	1F	00
	00	1F	00
	00	1F	00
	00	1B	00
	40	31	00
	60	60	00
1	00	1C	02
	00	1C	02
	00	1C	02
	00	08	03
	00	7E	01
	00	3E	00
	00	3F	00
	40	3F	00
	00	3E	00
	00	3E	00
	00	36	00
	00	36	00
	00	63	00

2



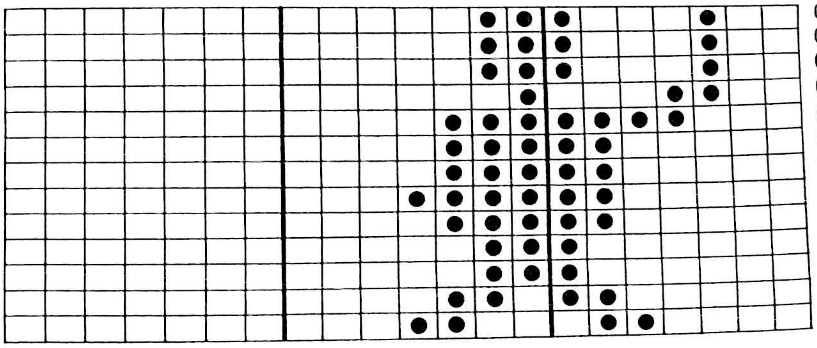
00	38	04
00	38	04
00	38	04
00	10	06
00	7C	03
00	7C	00
00	7C	00
00	7E	00
00	7C	00
00	38	00
00	38	00
00	6C	00
00	46	01

3



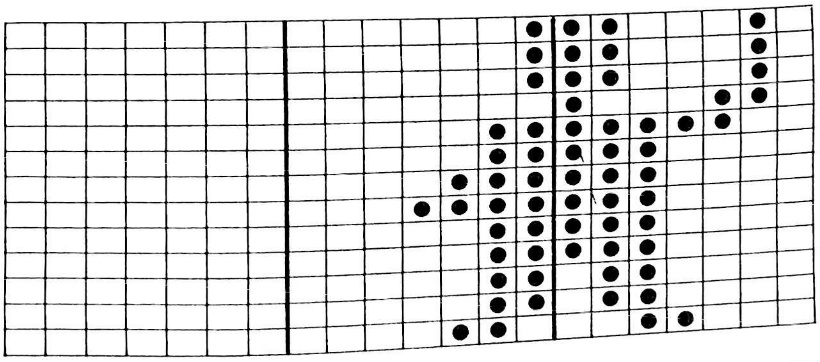
00	70	08
00	70	08
00	70	08
00	20	0C
00	78	07
00	78	01
00	78	01
00	78	01
00	78	01
00	70	00
00	70	00
00	70	00
00	70	00

4



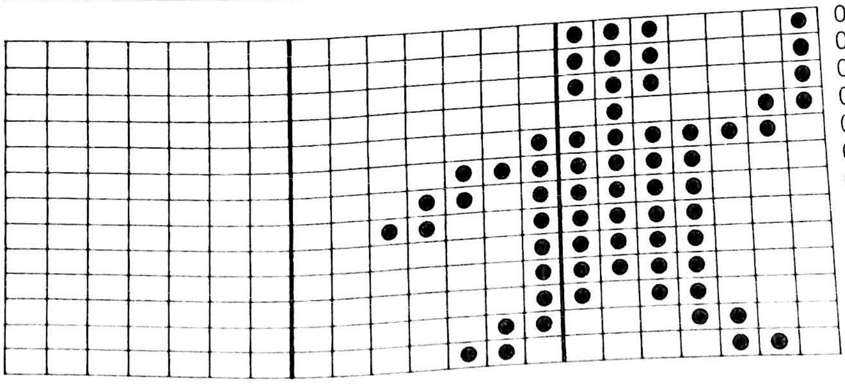
00	60	11
00	60	11
00	60	11
00	40	18
00	70	0F
00	70	03
00	70	03
00	78	03
00	70	03
00	60	01
00	60	01
00	30	03
00	18	06

5



00	40	23
00	40	23
00	40	23
00	00	31
00	60	1F
00	60	07
00	70	07
00	78	07
00	60	07
00	60	07
00	60	06
00	60	06
00	30	0C

6



00	00	47
00	00	47
00	00	47
00	00	62
00	40	3F
00	70	0F
00	58	0F
00	4C	0F
00	40	0F
00	40	0F
00	40	0D
00	60	18
00	30	30

At first glance, it might seem that these shape tables violate the rule of having an extra shape byte in the direction of movement. However, if the shape extends only one bit into the last byte, this is okay because there is room for all seven shapes in the last byte and an extra shape byte is not needed (see shape 6). We could have drawn the shapes over to the left, thus presenting the more usual type of shape tables, but the reason for not doing so is that drawing shapes this way makes it easier to align the fired bullet with the upraised arm, as we'll see in the next chapter.

The principle of internal animation is simple, but the application often is not because greater demands are placed on the artistic talents of the programmer. Even the crude animation of Program 5-5 required much time drawing and redrawing until I could stop the arms from flapping and keep the legs from placing themselves in anatomically impossible positions. Thank goodness for shape tables—they make this kind of tinkering much easier than if the shape bytes were dispersed throughout the draw routines.

You may envy, and with good reason, the type of internal animation found in some commercial game programs. I'm thinking specifically of Olympic Decathlon, which displays athletes running, jumping hurdles, throwing the javelin, and pole vaulting in exquisite silhouettes. These shapes almost surely were derived from photographs of athletes in action and transferred to the computer screen by talented artists, perhaps working with graphic utility programs on mainframe computers in Apple II simulation mode. But don't despair. I myself, devoid of the tiniest artistic talent, have successfully transferred complicated shapes from photographs by tracing the shape onto graph paper and filling in the dots. I even managed, at one time, to write a program displaying unicorns galloping across the screen, complete with heads bobbing and tails flapping. If I can do this, there is hope for anyone.


```
]PROGRAM 5-5
:ASM

                *1 SHAPE HORIZONTAL - INTERNAL ANIMATION
                *2 BYTES WIDE,13 LINES DEEP
                         ORG  $6000
6000: 4C 3F 60           JMP  PGM
                LINE     DS   1
                LINEA    DS   1
                BYTE     DS   1
                DEPTH    DS   1
                XCOUNT   DS   1
                SHPNO    DS   1
                DELAY    DS   1
                TEMP     DS   39
                GRAPHICS =    $C050
                MIXOFF   =    $C052
                HIRES    =    $C057
                PAGE1    =    $C054
                HIGH     =    $1B
                LOW      =    $1A
```

```
                    WAIT     =    $FCA8
                    *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                    *CONTINUE FOR ALL 7 SHAPES
6031: 28            SHPADR   DFB  #<SHAPE1
6032: 61                     DFB  #>SHAPE1
6033: 4F                     DFB  #<SHAPE2
6034: 61                     DFB  #>SHAPE2
6035: 76                     DFB  #<SHAPE3
6036: 61                     DFB  #>SHAPE3
6037: 9D                     DFB  #<SHAPE4
6038: 61                     DFB  #>SHAPE4
6039: C4                     DFB  #<SHAPE5
603A: 61                     DFB  #>SHAPE5
603B: EB                     DFB  #<SHAPE6
603C: 61                     DFB  #>SHAPE6
603D: 12                     DFB  #<SHAPE7
603E: 62                     DFB  #>SHAPE7
603F: AD 50 C0      PGM      LDA  GRAPHICS      ;HIRES,P.1
6042: AD 52 C0               LDA  MIXOFF
6045: AD 57 C0               LDA  HIRES
6048: AD 54 C0               LDA  PAGE1
604B: A9 00                  LDA  #$00          ;CLEAR SCREEN 1
604D: 85 1A                  STA  LOW
604F: A9 20                  LDA  #$20
6051: 85 1B                  STA  HIGH
6053: A0 00         CLR1     LDY  #$00
6055: A9 00                  LDA  #$00
6057: 91 1A         CLR      STA  (LOW),Y
6059: C8                     INY
605A: D0 FB                  BNE  CLR
605C: E6 1B                  INC  HIGH
605E: A5 1B                  LDA  HIGH
6060: C9 40                  CMP  #$40
6062: 90 EF                  BLT  CLR1
6064: A9 60                  LDA  #$60          ;LOAD DELAY
6066: 8D 09 60               STA  DELAY
6069: A2 B7                  LDX  #$B7          ;DRAW LINE
606B: A0 00                  LDY  #$00
606D: BD 39 62               LDA  HI,X
6070: 85 1B                  STA  HIGH
6072: BD F9 62               LDA  LO,X
6075: 85 1A                  STA  LOW
6077: A9 7F                  LDA  #$7F
6079: 91 1A         LN       STA  (LOW),Y
607B: C8                     INY
607C: C0 27                  CPY  #$27
607E: 90 F9                  BLT  LN
                    ********** MAIN PROGRAM **********
6080: 20 AE 60      START    JSR  INITIAL       ;SET INITIAL BYTE, LINE, DEPTH
6083: A9 00         START1   LDA  #$00          ;FIRST SHAPE NUMBER
6085: 8D 08 60               STA  SHPNO
6088: 20 C2 60      START2   JSR  LOADSHP       ;LOAD SHAPE INTO TEMP
608B: 20 DE 60               JSR  DRAW          ;DRAW
608E: AD 09 60               LDA  DELAY         ;DELAY
6091: 20 A8 FC               JSR  WAIT
6094: 20 DE 60               JSR  DRAW          ;ERASE
6097: EE 08 60               INC  SHPNO         ;NEXT SHAPE NUMBER
609A: AD 08 60               LDA  SHPNO
609D: C9 07                  CMP  #$07          ;FINISHED ALL 7 SHAPES?
609F: 90 E7                  BLT  START2        ;IF NO, CONTINUE WITH NEXT SHAPE
60A1: EE 05 60               INC  BYTE          ;IF YES, NEXT BYTE
```

```
60A4: AD 05 60          LDA  BYTE
60A7: C9 26             CMP  #$26          ;END OF SCREEN?
60A9: 90 D8             BLT  START1        ;IF NO, CONTINUE DRAW
60AB: 4C 80 60          JMP  START         ;IF YES, START OVER
             ********** SUBROUTINES **********
60AE: A9 00    INITIAL  LDA  #$00
60B0: 8D 05 60          STA  BYTE
60B3: A9 AA             LDA  #$AA
60B5: 8D 03 60          STA  LINE
60B8: 8D 04 60          STA  LINEA
60BB: 18                CLC
60BC: 69 0D             ADC  #$0D          ;DEPTH OF SHAPE
60BE: 8D 06 60          STA  DEPTH
60C1: 60                RTS
             **********************
60C2: AD 08 60 LOADSHP  LDA  SHPNO         ;LOAD SHAPE INTO TEMP
60C5: 0A                ASL
60C6: AA                TAX
60C7: BD 31 60          LDA  SHPADR,X
60CA: 85 1A             STA  LOW
60CC: BD 32 60          LDA  SHPADR+1,X
60CF: 85 1B             STA  HIGH
60D1: A0 00             LDY  #$00
60D3: B1 1A    LOADSHP1 LDA  (LOW),Y
60D5: 99 0A 60          STA  TEMP,Y
60D8: C8                INY
60D9: C0 27             CPY  #$27
60DB: 90 F6             BLT  LOADSHP1
60DD: 60                RTS
             **********************
60DE: A9 00    DRAW     LDA  #$00
60E0: 8D 07 60          STA  XCOUNT
60E3: AC 05 60 DRAW1    LDY  BYTE
60E6: AE 03 60          LDX  LINE
60E9: BD 39 62          LDA  HI,X
60EC: 85 1B             STA  HIGH
60EE: BD F9 62          LDA  LO,X
60F1: 85 1A             STA  LOW
60F3: AE 07 60          LDX  XCOUNT
60F6: B1 1A             LDA  (LOW),Y
60F8: 5D 0A 60          EOR  TEMP,X
60FB: 91 1A             STA  (LOW),Y
60FD: C8                INY
60FE: B1 1A             LDA  (LOW),Y
6100: 5D 0B 60          EOR  TEMP+1,X
6103: 91 1A             STA  (LOW),Y
6105: C8                INY
6106: B1 1A             LDA  (LOW),Y
6108: 5D 0C 60          EOR  TEMP+2,X
610B: 91 1A             STA  (LOW),Y
610D: EE 07 60          INC  XCOUNT
6110: EE 07 60          INC  XCOUNT
6113: EE 07 60          INC  XCOUNT
6116: EE 03 60          INC  LINE
6119: AD 03 60          LDA  LINE
611C: CD 06 60          CMP  DEPTH
611F: 90 C2             BLT  DRAW1
6121: AD 04 60          LDA  LINEA
6124: 8D 03 60          STA  LINE          ;RESET LINE FOR NEXT CYCLE
6127: 60                RTS
6128: 00 0E 01 SHAPE1   HEX  000E01000E01000E01
```

```
612B: 00 0E 01 00 0E 01
6131: 00 44 01   141          HEX 004401007F00601F00
6134: 00 7F 00 60 1F 00
613A: 30 1F 00   142          HEX 301F00181F00001F00
613D: 18 1F 00 00 1F 00
6143: 00 1F 00   143          HEX 001F00001B00403100
6146: 00 1B 00 40 31 00
614C: 60 60 00   144          HEX 606000
614F: 00 1C 02   145   SHAPE2 HEX 001C02001C02001C02
6152: 00 1C 02 00 1C 02
6158: 00 08 03   146          HEX 000803007E01003E00
615B: 00 7E 01 00 3E 00
6161: 00 3F 00   147          HEX 003F00403F00003E00
6164: 40 3F 00 00 3E 00
616A: 00 3E 00   148          HEX 003E00003600003600
616D: 00 36 00 00 36 00
6173: 00 63 00   149          HEX 006300
6176: 00 38 04   150   SHAPE3 HEX 003804003804003804
6179: 00 38 04 00 38 04
617F: 00 10 06   151          HEX 001006007C03007C00
6182: 00 7C 03 00 7C 00
6188: 00 7C 00   152          HEX 007C00007E00007C00
618B: 00 7E 00 00 7C 00
6191: 00 38 00   153          HEX 003800003800006C00
6194: 00 38 00 00 6C 00
619A: 00 46 01   154          HEX 004601
619D: 00 70 08   155   SHAPE4 HEX 007008007008007008
61A0: 00 70 08 00 70 08
61A6: 00 20 0C   156          HEX 00200C007807007801
61A9: 00 78 07 00 78 01
61AF: 00 78 01   157          HEX 007801007801007801
61B2: 00 78 01 00 78 01
61B8: 00 70 00   158          HEX 007000007000007000
61BB: 00 70 00 00 70 00
61C1: 00 70 00   159          HEX 007000
61C4: 00 60 11   160   SHAPE5 HEX 006011006011006011
61C7: 00 60 11 00 60 11
61CD: 00 40 18   161          HEX 00401800700F007003
61D0: 00 70 0F 00 70 03
61D6: 00 70 03   162          HEX 007003007803007003
61D9: 00 78 03 00 70 03
61DF: 00 60 01   163          HEX 006001006001003003
61E2: 00 60 01 00 30 03
61E8: 00 18 06   164          HEX 001806
61EB: 00 40 23   165   SHAPE6 HEX 004023004023004023
61EE: 00 40 23 00 40 23
61F4: 00 00 31   166          HEX 00003100601F006007
61F7: 00 60 1F 00 60 07
61FD: 00 70 07   167          HEX 007007007807006007
6200: 00 78 07 00 60 07
6206: 00 60 07   168          HEX 006007006006006006
6209: 00 60 06 00 60 06
620F: 00 30 0C   169          HEX 00300C
6212: 00 00 47   170   SHAPE7 HEX 000047000047000047
6215: 00 00 47 00 00 47
621B: 00 00 62   171          HEX 00006200403F00700F
621E: 00 40 3F 00 70 0F
6224: 00 58 0F   172          HEX 00580F004C0F00400F
6227: 00 4C 0F 00 40 0F
622D: 00 40 0F   173          HEX 00400F00400D006018
6230: 00 40 0D 00 60 18
```

```
6236: 00 30 30  174              HEX  003030
                         HI
                         LO
953 bytes
Symbol table - numerical order:
   LOW      =$1A       HIGH     =$1B       LINE     =$6003    LINEA    =$6004
   BYTE     =$6005     DEPTH    =$6006     XCOUNT   =$6007    SHPNO    =$6008
   DELAY    =$6009     TEMP     =$600A     SHPADR   =$6031    PGM      =$603F
   CLR1     =$6053     CLR      =$6057     LN       =$6079    START    =$6080
   START1   =$6083     START2   =$6088     INITIAL  =$60AE    LOADSHP  =$60C2
   LOADSHP1=$60D3      DRAW     =$60DE     DRAW1    =$60E3    SHAPE1   =$6128
   SHAPE2   =$614F     SHAPE3   =$6176     SHAPE4   =$619D    SHAPE5   =$61C4
   SHAPE6   =$61EB     SHAPE7   =$6212     HI       =$6239    LO       =$62F9
   GRAPHICS=$C050      MIXOFF   =$C052     PAGE1    =$C054    HIRES    =$C057
   WAIT     =$FCA8
```

One final note before leaving this chapter. The line drawing routine in Program 5-5 (lines 55-65) works okay, but there is a faster way to do it. Consider the routine from the program:

```
        LDA #$7F
LN      STA (LOW),Y
        INY
        CPY #$27
        BLT LN
```

Now consider the following routine, which draws the same line:

```
        LDY #$27
        LDA #$7F
LN      STA (LOW),Y
        DEY
        BPL LN
```

BPL (Branch on PLus) executes a branch if the result of an operation is in the range of #$00 to #$7F. Thus, the branch is taken until Y is decremented to #$FF. In both cases, a line is drawn from screen byte positions #$00 to #$27, but in the second case, the loop is shorter by one instruction because no comparison is done within the loop. This routine, however, while faster, is not as versatile as the first for two reasons: first, it works only if we wish a register to go to zero (or to #$01 if we use BNE LN)—it cannot be used to draw the line from screen byte #$27 to #$05, for example, and second, the loop cannot be initialized with a value greater than #$7F—this routine for example would not work in the LOADSHP subroutine if the shape were larger than #$7F. Nevertheless, I mention it because it is an example of good programming technique and while I use the more versatile routine in the programs in this book, the second routine should be kept in mind for program optimization where applicable.

Congratulations! You have now learned the basic principles of hi-res drawing and animation. With this knowledge you now should be able to draw any shape and move it around the screen, even in complicated paths (by changing line positions and screen bytes at the same time instead of just one or the other). This knowledge in itself provides you with a powerful tool for a wide variety of applications. The remaining chapters in Part One will deal with aspects of game design and construction together with techniques of more general applicablity such as animating multiple shapes discussed in Chapter 6. The chapters in Part Two discuss other aspects of hi-res animation applicable to both game programs and any other type of program where hi-res animation would be useful.

6

Paddle and Joystick Controls and Multiple Shapes

How's this for a bit of twaddle—
Try moving a duck shape with a paddle.
If your hand is unsteady
The duck will, are you ready?
Move with a quite pronounced waddle.

What would a game be without a joystick or paddles? A game without a joystick or paddles. Take my keyboard, please. Seriously folks, some games use the keyboard to control shape movement and initiate actions such as shooting bullets, but joystick or paddle controls are much easier to use and are more entertaining—that's why they exist and why most game programs utilize them. In this chapter we're going to see how to use these hand controls to control vertical and horizontal movement and how to use the "firing" buttons. We're also going to discuss the not insignificant problem of how to display two different moving shapes at the same time. Most of the routines in this chapter will be used in the final game program.

PADDLE AND JOYSTICK CONTROLS

Paddles have rotary knobs and come in sets of two, paddle 0 and paddle 1, each with its own "firing" button. A joystick combines both paddles into a single instrument—the two joystick buttons are equivalent to the paddle buttons. Thus, joysticks and paddles can be used interchangeably although finer control is afforded by paddles. By choosing the appropriate instructions, one can access either paddle 0 or paddle 1 (equivalent to joystick left-right or forward-back) or either button. When using a joystick, it doesn't make any difference which button is chosen but with paddles, one should choose the button appropriate to the paddle—using paddle 0 with button 1 or paddle 1 with button 0 would require a certain amount of dexterity certain not to be appreciated by the program's user.

"Reading" a paddle or joystick (i.e., determining the position of the knob or stick) fortunately is made easy by accessing a built-in Apple II subroutine located at memory address $FB1E, which is labelled PREAD in our programs. The number of the hand control you want to access is placed in the X register and a

JSR PREAD then returns a number from 0 to 255 (#$00 to #$FF) in the Y register, the particular number depending on the hand control position. Thus:

```
LDX #$00      Read paddle 0 (stick left-right)
JSR PREAD     Returns 0-255 in Y register

LDX #$01      Read paddle 1 (stick forward-back)
JSR PREAD     Returns 0-255 in Y register
```

The number in Y can then be manipulated to select screen byte for horizontal movement or line for vertical movement (more about this soon).

To test whether a button is pressed or not requires only reading soft switches, $C061 for button 0 and $C062 for button 1. In conjunction with the opcode BMI (Branch on MInus), the branch is taken if the button is pressed and not taken if the button is not pressed. Thus:

```
      LDA $C061
      BMI CONT    If button 0 is pressed, branch to CONT
      RTS
CONT  JSR DRAW

      LDA $C062
      BMI CONT    If button 1 is pressed, branch to CONT
      RTS
CONT  JSR DRAW
```

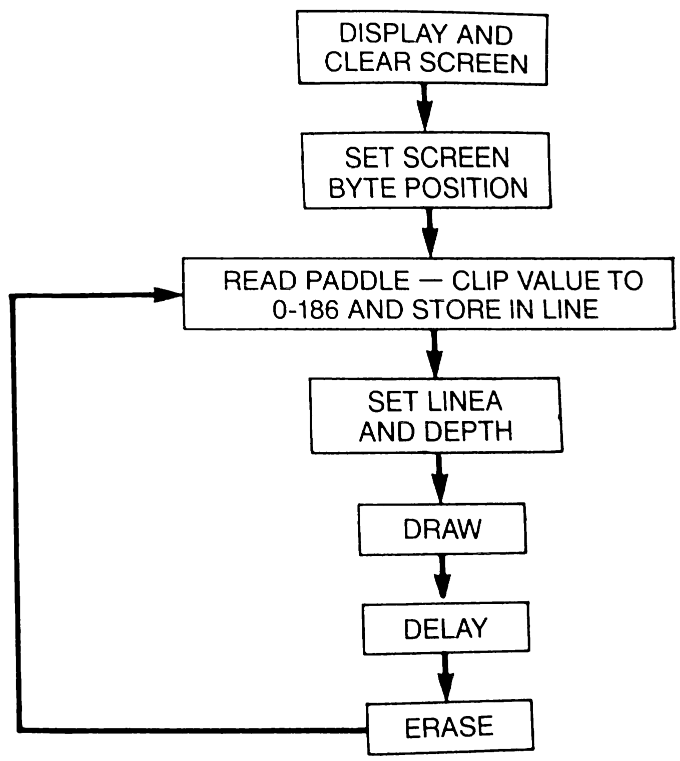

That's all there is to it! Let's see now how we can adapt these routines to moving shapes around the screen (for convenience, from now on I will use the term paddle to refer to both paddle and joystick).

PADDLE CONTROL OF VERTICAL MOVEMENT

The next program (Program 6-1) is essentially identical to Program 4-2 (moving a spaceship vertically), except now we're going to control the spaceship's vertical movement by paddle 1. Let's examine Program 6-1 in some detail.

First, in the beginning of the program, we define $FB1E as PREAD. Then, in the MAIN PROGRAM, we go to the INITIAL subroutine to set the screen byte. We do not set the line position here because LINE will be determined by the paddle position and we do not set DEPTH here either, as DEPTH depends on LINE. The next instruction sends the program to the PDLE subroutine—it is here that we read paddle 1 and return a value of 0-255 in the Y register (LDX #$01, JSR PREAD), and we're going to use the value in Y to set the line position. However, as there are only 192 screen lines (0-191), we first have to clip the maximum value in Y to keep the shape on the screen. Because the shape is 6 lines deep and is drawn from the top line down, the maximum line position we want in LINE is 186 (#$BA)—the shape will then be drawn from lines 186 to 191. The instructions in lines 63 to 67 accomplish the clipping. We compare the value in Y to 187 (#$BB) and if it less than this, we store the value in Y in LINE. If it is equal to or greater than 187, we store the value 186 (#$BA) in Y and then store Y in LINE (LDA #$BA, TAY [Transfer A to Y], STY LINE). Thus, no matter what the paddle position, LINE will not contain a value greater than 186 and this keeps the shape on the screen.

We then go back to the MAIN PROGRAM and jump to the DEP subroutine which stores LINE in LINEA and also sets DEPTH—remember that while the shape depth is a constant, the value in DEPTH depends on the value in LINE. Back in the MAIN PROGRAM, we draw the shape with JSR DRAW, delay, and erase with JSR DRAW (we're using the DRAW-ERASE protocol). The next instruction sends the program back to PADDLE for another paddle read and we continue in this loop, continually updating LINE from the paddle position.




```
]PROGRAM 6-1
:ASM
                *ONE SHAPE VERTICAL CONTROLLED BY PADDLE*
                *****************************
                *SHAPE IS 1 BYTE WIDE BY 6 BYTES DEEP
                ****************************
                         ORG   $6000
6000: 4C 09 60           JMP   PGM
                XCOUNT   DS    1
                BYTE     DS    1
                LINE     DS    1
                LINEA    DS    1
                DEPTH    DS    1
                DELAY    DS    1
                GRAPHICS =     $C050
                MIXOFF   =     $C052
                HIRES    =     $C057
                PAGE1    =     $C054
                HIGH     =     $1B
                LOW      =     $1A
                WAIT     =     $FCA8
                PREAD    =     $FB1E
6009: AD 50 C0  PGM      LDA   GRAPHICS   ;HIRES,P.1
600C: AD 52 C0           LDA   MIXOFF
```

```
600F: AD 57 C0              LDA  HIRES
6012: AD 54 C0              LDA  PAGE1
6015: A9 00                 LDA  #$00         ;CLEAR SCREEN 1
6017: 85 1A                 STA  LOW
6019: A9 20                 LDA  #$20
601B: 85 1B                 STA  HIGH
601D: A0 00        CLR1     LDY  #$00
601F: A9 00                 LDA  #$00
6021: 91 1A        CLR      STA  (LOW),Y
6023: C8                    INY
6024: D0 FB                 BNE  CLR
6026: E6 1B                 INC  HIGH
6028: A5 1B                 LDA  HIGH
602A: C9 40                 CMP  #$40
602C: 90 EF                 BLT  CLR1
602E: A9 40                 LDA  #$40         ;LOAD TIME DELAY
6030: 8D 08 60              STA  DELAY
                   ********** MAIN PROGRAM **********
6033: 20 4B 60              JSR  INITIAL      ;SET SCREEN BYTE
6036: 20 5E 60     PADDLE   JSR  PDLE         ;READ PADDLE 1
6039: 20 51 60              JSR  DEP          ;SET DEPTH
603C: 20 6E 60              JSR  DRAW         ;DRAW
603F: AD 08 60              LDA  DELAY
6042: 20 A8 FC              JSR  WAIT         ;DELAY
6045: 20 6E 60              JSR  DRAW         ;ERASE
6048: 4C 36 60              JMP  PADDLE       ;READ PADDLE AGAIN
                   ********** SUBROUTINES **********
604B: A9 10        INITIAL  LDA  #$10
604D: 8D 04 60              STA  BYTE         ;SET STARTING BYTE
6050: 60                    RTS
                   ***********************
6051: AD 05 60     DEP      LDA  LINE         ;SET DEPTH
6054: 8D 06 60              STA  LINEA
6057: 18                    CLC
6058: 69 06                 ADC  #$06
605A: 8D 07 60              STA  DEPTH
605D: 60                    RTS
                   **********************
605E: A2 01        PDLE     LDX  #$01         ;READ PADDLE 1
6060: 20 1E FB              JSR  PREAD        ;0-255 IN Y
6063: C0 BB                 CPY  #$BB         ;CLIP TO 0-186
6065: 90 03                 BLT  CONT
6067: A9 BA                 LDA  #$BA
6069: A8                    TAY
606A: 8C 05 60     CONT     STY  LINE         ;0-186 IN LINE
606D: 60                    RTS
                   *********************
606E: A9 00        DRAW     LDA  #$00
6070: 8D 03 60              STA  XCOUNT       ;ZERO XCOUNT
6073: AC 04 60     DRAW1    LDY  BYTE         ;LOAD BYTE
6076: AE 05 60              LDX  LINE         ;LOAD LINE
6079: BD A8 60              LDA  HI,X         ;LOAD LINE ADDRESS INTO HIGH,LOW
607C: 85 1B                 STA  HIGH
607E: BD 68 61              LDA  LO,X
6081: 85 1A                 STA  LOW
6083: AE 03 60              LDX  XCOUNT       ;LOAD X WITH XCOUNT
6086: B1 1A                 LDA  (LOW),Y      ;GET BYTE FROM SCREEN
6088: 5D A2 60              EOR  SHAPE,X      ;EOR BYTE FROM SHAPE ADDRESS+X
608B: 91 1A                 STA  (LOW),Y      ;PLOT BYTE
608D: EE 03 60              INC  XCOUNT
6090: EE 05 60              INC  LINE         ;NEXT LINE
```

```
6093: AD 05 60   84          LDA  LINE
6096: CD 07 60   85          CMP  DEPTH        ;FINISH SHAPE?
6099: 90 D8      86          BLT  DRAW1        ;IF NO, DRAW NEXT LINE
609B: AD 06 60   87          LDA  LINEA        ;IF YES, RESET LINE AND
609E: 8D 05 60   88          STA  LINE            DRAW NEXT CYCLE
60A1: 60         89          RTS
60A2: 08 1C 22   90   SHAPE  HEX  081C223E227F ;SHAPE TABLE
60A5: 3E 22 7F
                      HI
                      LO

552 bytes

Symbol table - numerical order:

  LOW      =$1A     HIGH      =$1B     XCOUNT  =$6003   BYTE    =$6004
  LINE     =$6005   LINEA     =$6006   DEPTH   =$6007   DELAY   =$6008
  PGM      =$6009   CLR1      =$601D   CLR     =$6021   PADDLE  =$6036
  INITIAL  =$604B   DEP       =$6051   PDLE    =$605E   CONT    =$606A
  DRAW     =$606E   DRAW1     =$6073   SHAPE   =$60A2   HI      =$60A8
  LO       =$6168   GRAPHICS=$C050     MIXOFF  =$C052   PAGE1   =$C054
  HIRES    =$C057   PREAD     =$FB1E   WAIT    =$FCA8
```


PADDLE CONTROL OF HORIZONTAL MOVEMENT

In contrast to paddle control of vertical movement where we specify a particular screen byte position and use a paddle read to select the line position, paddle control of horizontal movement involves specifying a particular line position and using the paddle read to select the screen byte position. However, as you might suspect, things are not always that simple with horizontal movement. Remember that in horizontal movement, each screen byte can contain one of seven shapes. Therefore, we not only have to specify the screen byte position by a paddle read, but also which shape is to be drawn. Specifying shapes was relatively easy in previous programs because we started with the first shape and then accessed the other shapes sequentially. With a paddle read, shape and screen byte selection is accomplished by the use of look-up tables. We'll see how this is done in the next program (Program 6-2), which is based on Program 5-5, the horizontal movement of a person shape with internal animation (Program 6-2 will be incorporated into the final game program).

In Program 6-2, we're going to use the same shape tables and the line for the person to walk on as Program 5-5 and we're going to control horizontal movement by paddle 0. In the MAIN PROGRAM, we first go to the INITIAL subroutine to set LINE and DEPTH. Then we jump to the PDLE subroutine, which reads paddle 0 and returns a value of 0-255 in the Y register.

We first want to convert the value in Y to a screen byte position. We do this by the instruction LDA BYTETBL,Y (line 85) where BYTETBL is a table consisting of 37 lines of 7 bytes each, 7 #$00's, 7 #$01's, 7 #$02's, etc., up to 7 #$24's. A screen byte from 0 to 36 is selected, depending on the value in Y; i.e.,

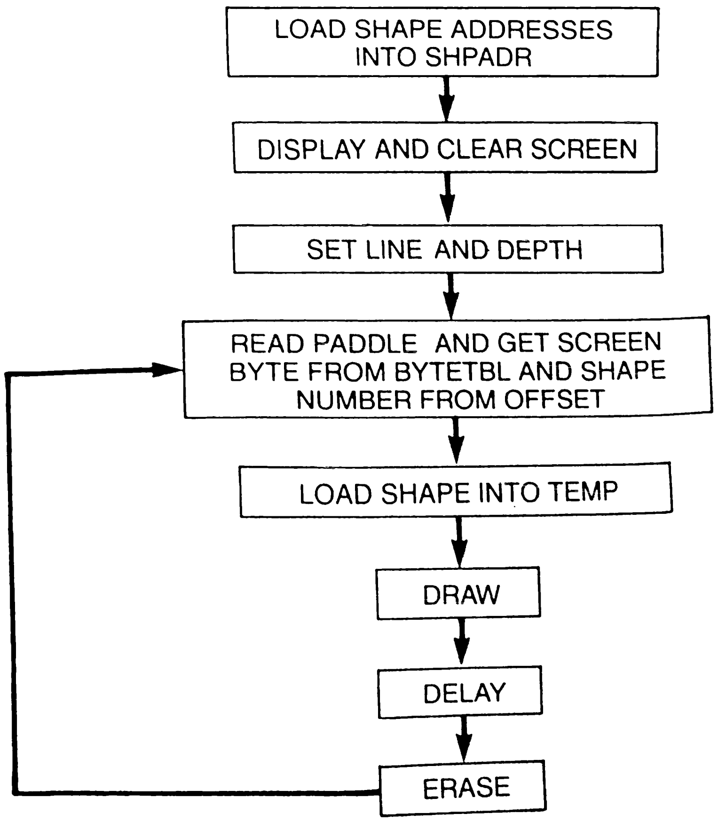
Value in Y	*Screen Byte*
0-6	0
7-13	1
14-20	2
21-27	3
.	.
.	.
245-251	35
252-255	36

The screen byte obtained is then stored in HORIZ (line 86), which will be used in the draw routine to denote the screen byte position. Note that we are accessing only 37 (0-36) screen bytes even though 40 (0-39) are available. This is because Y can contain a maximum value of 255 and, to access all 40 screen bytes, a value of 280 would be needed (40 X 7). We could make the BYTETBL shorter by storing, for example, 6 bytes per line for 40 lines, but there is a reason for having 7 bytes per line as we'll soon see (if you think this is related to 7 shapes per screen byte, you're right). There are ways to access the extra bytes at the end of the screen, but this would present an unnecessary complication as you would see by running the program—the inability of the shape to move into the last few bytes at the end of the screen is hardly noticeable. The shape can be centered by adding an offset to the byte position so that, for example, bytes 1-37 would be selected, but this is not necessary in our program.

Now that we have the screen byte, we want to specify which of the seven shapes to draw. This is accomplished by using another look-up table. The instruction in line 87 (LDA OFFSET,Y) accesses the table labelled OFFSET which, like BYTETBL, also contains 37 lines of 7 bytes each. But, here, each line contains the same bytes, #$00 to #$06. The value in Y selects a shape number, 0 to 6; i.e.:

Value in Y	*Screen Byte*	*Shape Number*
0	0	0
1	0	1
2	0	2
3	0	3
4	0	4
5	0	5
6	0	6
7	1	0
8	1	1
9	1	2
10	1	3
11	1	4
12	1	5
13	1	6
14	2	0
.	.	.
.	.	.
252	36	0
253	36	1
254	36	2
255	36	3

The instruction LDA OFFSET,Y loads the Accumulator with a shape number and the rest of the PDLE subroutine loads the shape into TEMP using the same instructions we've seen in Chapter 5. The program then draws the shape, delays, erases, and loops back to PADDLE to update the horizontal position continually. The DRAW routine is the same as in previous programs except that Y is loaded with the value in HORIZ instead of BYTE (BYTE is simply not used in this program).



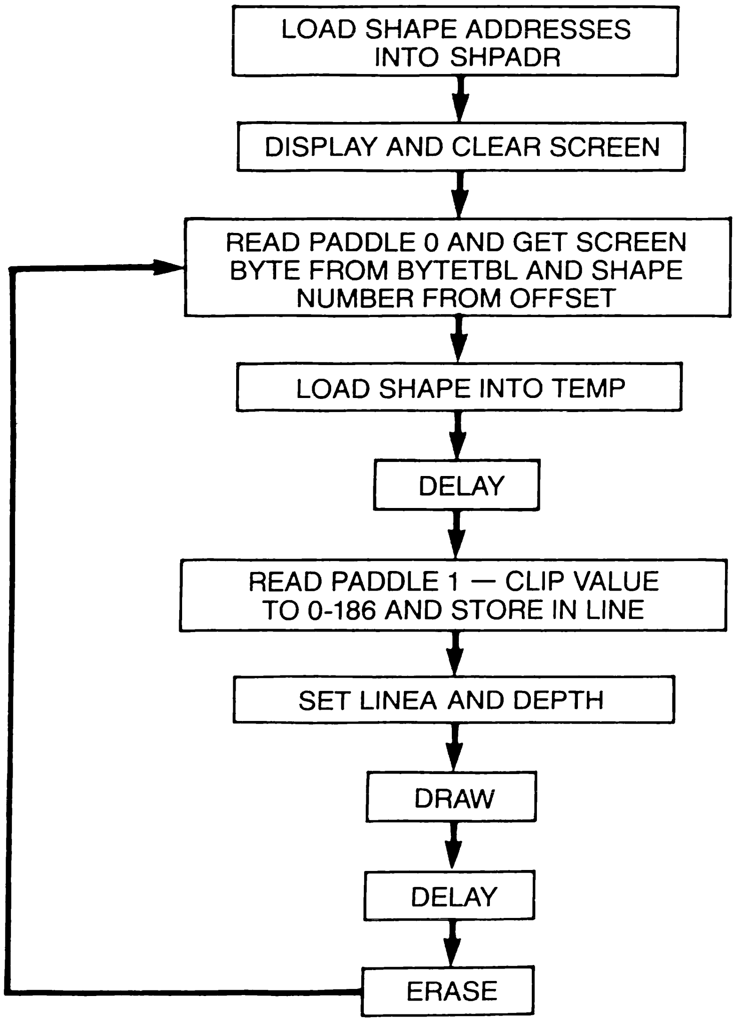

```
]PROGRAM 6-2
:ASM
                *PADDLE OR JOYSTICK CONTROL OF HORIZONTAL MOVEMENT
                *2 BYTES WIDE, 13 LINES DEEP
                         ORG  $6000
6000: 4C 3E 60           JMP  PGM
                LINE     DS   1
                LINEA    DS   1
                DEPTH    DS   1
                HORIZ    DS   1
                XCOUNT   DS   1
                DELAY    DS   1
                TEMP     DS   39
                GRAPHICS =    $C050
                MIXOFF   =    $C052
                HIRES    =    $C057
                PAGE1    =    $C054
                HIGH     =    $1B
                LOW      =    $1A
                WAIT     =    $FCA8
                PREAD    =    $FB1E
                *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                *CONTINUE FOR ALL 7 SHAPES
```

```
6030: 14             SHPADR   DFB   #<SHAPE1
6031: 61                      DFB   #>SHAPE1
6032: 3B                      DFB   #<SHAPE2
6033: 61                      DFB   #>SHAPE2
6034: 62                      DFB   #<SHAPE3
6035: 61                      DFB   #>SHAPE3
6036: 89                      DFB   #<SHAPE4
6037: 61                      DFB   #>SHAPE4
6038: B0                      DFB   #<SHAPE5
6039: 61                      DFB   #>SHAPE5
603A: D7                      DFB   #<SHAPE6
603B: 61                      DFB   #>SHAPE6
603C: FE                      DFB   #<SHAPE7
603D: 61                      DFB   #>SHAPE7
603E: AD 50 C0       PGM      LDA   GRAPHICS     ;HIRES,P.1
6041: AD 52 C0                LDA   MIXOFF
6044: AD 57 C0                LDA   HIRES
6047: AD 54 C0                LDA   PAGE1
604A: A9 00                   LDA   #$00         ;CLEAR SCREEN 1
604C: 85 1A                   STA   LOW
604E: A9 20                   LDA   #$20
6050: 85 1B                   STA   HIGH
6052: A0 00          CLR1     LDY   #$00
6054: A9 00                   LDA   #$00
6056: 91 1A          CLR      STA   (LOW),Y
6058: C8                      INY
6059: D0 FB                   BNE   CLR
605B: E6 1B                   INC   HIGH
605D: A5 1B                   LDA   HIGH
605F: C9 40                   CMP   #$40
6061: 90 EF                   BLT   CLR1
6063: A9 60                   LDA   #$60         ;LOAD DELAY
6065: 8D 08 60                STA   DELAY
6068: A2 B7                   LDX   #$B7         ;DRAW LINE
606A: A0 00                   LDY   #$00
606C: BD 2B 64                LDA   HI,X
606F: 85 1B                   STA   HIGH
6071: BD EB 64                LDA   LO,X
6074: 85 1A                   STA   LOW
6076: A9 7F                   LDA   #$7F
6078: 91 1A          LN       STA   (LOW),Y
607A: C8                      INY
607B: C0 27                   CPY   #$27
607D: 90 F9                   BLT   LN
                     ********** MAIN PROGRAM **********
607F: 20 94 60                JSR   INIT         ;SET LINE & DEPTH
6082: 20 A3 60       PADDLE   JSR   PDLE         ;READ PADDLE 0
6085: 20 CA 60                JSR   DRAW         ;DRAW
6088: AD 08 60                LDA   DELAY
608B: 20 A8 FC                JSR   WAIT         ;DELAY
608E: 20 CA 60                JSR   DRAW         ;ERASE
6091: 4C 82 60                JMP   PADDLE       ;READ PADDLE AGAIN
                     ********** SUBROUTINES **********
6094: A9 AA          INIT     LDA   #$AA
6096: 8D 03 60                STA   LINE
6099: 8D 04 60                STA   LINEA
609C: 18                      CLC
609D: 69 0D                   ADC   #$0D
609F: 8D 05 60                STA   DEPTH
60A2: 60                      RTS
                     ***********************
```

```
60A3: A2 00       PDLE    LDX  #$00
60A5: 20 1E FB            JSR  PREAD        ;READ PADDLE 0
60A8: B9 25 62            LDA  BYTETBL,Y    ;CONVERT TO SCREEN BYTE (0 - 36)
60AB: 8D 06 60            STA  HORIZ
60AE: B9 28 63            LDA  OFFSET,Y     ;GET SHAPE NUMBER
60B1: 0A                  ASL               ;LOAD SHAPE INTO TEMP
60B2: AA                  TAX
60B3: BD 30 60            LDA  SHPADR,X
60B6: 85 1A               STA  LOW
60B8: BD 31 60            LDA  SHPADR+1,X
60BB: 85 1B               STA  HIGH
60BD: A0 00               LDY  #$00
60BF: B1 1A       LOAD    LDA  (LOW),Y
60C1: 99 09 60            STA  TEMP,Y
60C4: C8                  INY
60C5: C0 27               CPY  #$27
60C7: 90 F6               BLT  LOAD
60C9: 60                  RTS
                  ************************
60CA: A9 00       DRAW    LDA  #$00
60CC: 8D 07 60            STA  XCOUNT
60CF: AE 03 60    DRAW1   LDX  LINE
60D2: AC 06 60            LDY  HORIZ
60D5: BD 2B 64            LDA  HI,X
60D8: 85 1B               STA  HIGH
60DA: BD EB 64            LDA  LO,X
60DD: 85 1A               STA  LOW
60DF: AE 07 60            LDX  XCOUNT
60E2: B1 1A               LDA  (LOW),Y
60E4: 5D 09 60            EOR  TEMP,X
60E7: 91 1A               STA  (LOW),Y
60E9: C8                  INY
60EA: B1 1A               LDA  (LOW),Y
60EC: 5D 0A 60            EOR  TEMP+1,X
60EF: 91 1A               STA  (LOW),Y
60F1: C8                  INY
60F2: B1 1A               LDA  (LOW),Y
60F4: 5D 0B 60            EOR  TEMP+2,X
      91 1A               STA  (LOW),Y
      EE 07 60            INC  XCOUNT
      EE 07 60            INC  XCOUNT
      EE 07 60            INC  XCOUNT
      EE 03 60            INC  LINE
      AD 03 60            LDA  LINE
      CD 05 60            CMP  DEPTH
      90 C2               BLT  DRAW1        ;RESET LINE
      AD 04 60            LDA  LINEA
      8D 03 60            STA  LINE
      60                  RTS               ;SHAPE TABLES
      00 0E 01    SHAPE1  HEX  000E01000E01000E01
      00 0E 01 00 0E 01
      00 44 01            HEX  004401007F00601F00
      00 7F 00 60 1F 00
      30 1F 00            HEX  301F00181F00001F00
      18 1F 00 00 1F 00
      00 1F 00            HEX  001F00001B00403100
      00 1B 00 40 31 00
      60 60 00            HEX  606000
      00 1C 02    SHAPE2  HEX  001C02001C02001C02
      00 1C 02 00 1C 02
      00 08 03            HEX  000803007E01003E00
```

```
       00 7E 01 00 3E 00
       00 3F 00  139                HEX  003F00403F00003E00
       40 3F 00 00 3E 00
       00 3E 00  140                HEX  003E00003600003600
       00 36 00 00 36 00
       00 63 00  141                HEX  006300
       00 38 04  142  SHAPE3        HEX  003804003804003804
       00 38 04 00 38 04
       00 10 06  143                HEX  001006007C03007C00
       00 7C 03 00 7C 00
       00 7C 00  144                HEX  007C00007E00007C00
       00 7E 00 00 7C 00
       00 38 00  145                HEX  003800003800006C00
       00 38 00 00 6C 00
       00 46 01  146                HEX  004601
       00 70 08  147  SHAPE4        HEX  007008007008007008
       00 70 08 00 70 08
       00 20 0C  148                HEX  00200C007807007801
       00 78 07 00 78 01
       00 78 01  149                HEX  007801007801007801
       00 78 01 00 78 01
       00 70 00  150                HEX  007000007000007000
       00 70 00 00 70 00
       00 70 00  151                HEX  007000
       00 60 11  152  SHAPE5        HEX  006011006011006011
       00 60 11 00 60 11
       00 40 18  153                HEX  00401800700F007003
       00 70 0F 00 70 03
       00 70 03  154                HEX  007003007803007003
       00 78 03 00 70 03
       00 60 01  155                HEX  006001006001003003
       00 60 01 00 30 03
       00 18 06  156                HEX  001806
       00 40 23  157  SHAPE6        HEX  004023004023004023
       00 40 23 00 40 23
       00 00 31  158                HEX  00003100601F006007
       00 60 1F 00 60 07
       00 70 07  159                HEX  007007007807006007
       00 78 07 00 60 07
61F2:  00 60 07  160                HEX  006007006006006006
61F5:  00 60 06 00 60 06
61FB:  00 30 0C  161                HEX  00300C
61FE:  00 00 47  162  SHAPE7        HEX  000047000047000047
6201:  00 00 47 00 00 47
6207:  00 00 62  163                HEX  00006200403F00700F
620A:  00 40 3F 00 70 0F
6210:  00 58 0F  164                HEX  00580F004C0F00400F
6213:  00 4C 0F 00 40 0F
6219:  00 40 0F  165                HEX  00400F00400D006018
621C:  00 40 0D 00 60 18
6222:  00 30 30  166                HEX  003030
6225:  00 00 00  167  BYTETBL       HEX  00000000000000
6228:  00 00 00 00
622C:  01 01 01  168                HEX  01010101010101
622F:  01 01 01 01
6233:  02 02 02  169                HEX  02020202020202
6236:  02 02 02 02
623A:  03 03 03  170                HEX  03030303030303
623D:  03 03 03 03
6241:  04 04 04  171                HEX  04040404040404
6244:  04 04 04 04
```

```
6248: 05 05 05  172           HEX  05050505050505
624B: 05 05 05 05
624F: 06 06 06  173           HEX  06060606060606
6252: 06 06 06 06
6256: 07 07 07  174           HEX  07070707070707
6259: 07 07 07 07
625D: 08 08 08  175           HEX  08080808080808
6260: 08 08 08 08
6264: 09 09 09  176           HEX  09090909090909
6267: 09 09 09 09
626B: 0A 0A 0A  177           HEX  0A0A0A0A0A0A0A
626E: 0A 0A 0A 0A
6272: 0B 0B 0B  178           HEX  0B0B0B0B0B0B0B
6275: 0B 0B 0B 0B
6279: 0C 0C 0C  179           HEX  0C0C0C0C0C0C0C
627C: 0C 0C 0C 0C
6280: 0D 0D 0D  180           HEX  0D0D0D0D0D0D0D
6283: 0D 0D 0D 0D
6287: 0E 0E 0E  181           HEX  0E0E0E0E0E0E0E
628A: 0E 0E 0E 0E
628E: 0F 0F 0F  182           HEX  0F0F0F0F0F0F0F
6291: 0F 0F 0F 0F
6295: 10 10 10  183           HEX  10101010101010
6298: 10 10 10 10
629C: 11 11 11  184           HEX  11111111111111
629F: 11 11 11 11
62A3: 12 12 12  185           HEX  12121212121212
62A6: 12 12 12 12
62AA: 13 13 13  186           HEX  13131313131313
62AD: 13 13 13 13
62B1: 14 14 14  187           HEX  14141414141414
62B4: 14 14 14 14
62B8: 15 15 15  188           HEX  15151515151515
62BB: 15 15 15 15
62BF: 16 16 16  189           HEX  16161616161616
62C2: 16 16 16 16
62C6: 17 17 17  190           HEX  17171717171717
62C9: 17 17 17 17
62CD: 18 18 18  191           HEX  18181818181818
62D0: 18 18 18 18
62D4: 19 19 19  192           HEX  19191919191919
62D7: 19 19 19 19
62DB: 1A 1A 1A  193           HEX  1A1A1A1A1A1A1A
62DE: 1A 1A 1A 1A
62E2: 1B 1B 1B  194           HEX  1B1B1B1B1B1B1B
62E5: 1B 1B 1B 1B
62E9: 1C 1C 1C  195           HEX  1C1C1C1C1C1C1C
62EC: 1C 1C 1C 1C
62F0: 1D 1D 1D  196           HEX  1D1D1D1D1D1D1D
62F3: 1D 1D 1D 1D
62F7: 1E 1E 1E  197           HEX  1E1E1E1E1E1E1E
62FA: 1E 1E 1E 1E
62FE: 1F 1F 1F  198           HEX  1F1F1F1F1F1F1F
6301: 1F 1F 1F 1F
6305: 20 20 20  199           HEX  20202020202020
6308: 20 20 20 20
630C: 21 21 21  200           HEX  21212121212121
630F: 21 21 21 21
6313: 22 22 22  201           HEX  22222222222222
6316: 22 22 22 22
631A: 23 23 23  202           HEX  23232323232323
```




```
]PROGRAM 6-3
:ASM
                 *JOYSTICK CONTROL OF HORIZONTAL & VERTICAL MOVEMENT
                 *2 BYTES WIDE, 13 LINES DEEP
                           ORG   $6000
6000: 4C 26 60             JMP   PGM
                 LINE      DS    1
                 LINEA     DS    1
                 DEPTH     DS    1
                 HORIZ     DS    1
                 XCOUNT    DS    1
                 DELAY     DS    1
                 TEMP      DS    15
                 GRAPHICS  =     $C050
                 MIXOFF    =     $C052
                 HIRES     =     $C057
                 PAGE1     =     $C054
                 HIGH      =     $1B
                 LOW       =     $1A
                 WAIT      =     $FCA8
                 PREAD     =     $FB1E
                 *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                 *CONTINUE FOR ALL 7 SHAPES
6018: F7         SHPADR    DFB   #<SHAPE1
6019: 60                   DFB   #>SHAPE1
```

```
601A: 06                       DFB  #<SHAPE2
601B: 61                       DFB  #>SHAPE2
601C: 15                       DFB  #<SHAPE3
601D: 61                       DFB  #>SHAPE3
601E: 24                       DFB  #<SHAPE4
601F: 61                       DFB  #>SHAPE4
6020: 33                       DFB  #<SHAPE5
6021: 61                       DFB  #>SHAPE5
6022: 42                       DFB  #<SHAPE6
6023: 61                       DFB  #>SHAPE6
6024: 51                       DFB  #<SHAPE7
6025: 61                       DFB  #>SHAPE7
6026: AD 50 C0       PGM       LDA  GRAPHICS    ;HIRES,P.1
6029: AD 52 C0                 LDA  MIXOFF
602C: AD 57 C0                 LDA  HIRES
602F: AD 54 C0                 LDA  PAGE1
6032: A9 00                    LDA  #$00        ;CLEAR SCREEN 1
6034: 85 1A                    STA  LOW
6036: A9 20                    LDA  #$20
6038: 85 1B                    STA  HIGH
603A: A0 00          CLR1      LDY  #$00
603C: A9 00                    LDA  #$00
603E: 91 1A          CLR       STA  (LOW),Y
6040: C8                       INY
6041: D0 FB                    BNE  CLR
6043: E6 1B                    INC  HIGH
6045: A5 1B                    LDA  HIGH
6047: C9 40                    CMP  #$40
6049: 90 EF                    BLT  CLR1
604B: A9 60                    LDA  #$60        ;LOAD DELAY
604D: 8D 08 60                 STA  DELAY
                     ********** MAIN PROGRAM **********
6050: 20 6A 60       PADDLE    JSR  PDLE0
6053: A9 20                    LDA  #$20
6055: 20 A8 FC                 JSR  WAIT
6058: 20 91 60                 JSR  PDLE1
605B: 20 AD 60                 JSR  DRAW
605E: AD 08 60                 LDA  DELAY
6061: 20 A8 FC                 JSR  WAIT
6064: 20 AD 60                 JSR  DRAW
6067: 4C 50 60                 JMP  PADDLE
                     ********** SUBROUTINES **********
606A: A2 00          PDLE0     LDX  #$00
606C: 20 1E FB                 JSR  PREAD       ;READ PADDLE 0
606F: B9 60 61                 LDA  BYTETBL,Y   ;CONVERT TO SCREEN BYTE (0 - 36)
6072: 8D 06 60                 STA  HORIZ
6075: B9 63 62                 LDA  OFFSET,Y    ;GET SHAPE NUMBER
6078: 0A                       ASL              ;LOAD SHAPE INTO TEMP
6079: AA                       TAX
607A: BD 18 60                 LDA  SHPADR,X
607D: 85 1A                    STA  LOW
607F: BD 19 60                 LDA  SHPADR+1,X
6082: 85 1B                    STA  HIGH
6084: A0 00                    LDY  #$00
6086: B1 1A          LOAD      LDA  (LOW),Y
6088: 99 09 60                 STA  TEMP,Y
608B: C8                       INY
608C: C0 0F                    CPY  #$0F
608E: 90 F6                    BLT  LOAD
6090: 60                       RTS
6091: A2 01          PDLE1     LDX  #$01
```

pressed. Also, the program is designed so that only one bullet can be fired at a time; i.e., a bullet on the screen must go off before the next one can be drawn (the program doesn't have to have this feature but what the heck, why not). We accomplish this by reserving a memory location labelled BULON and loading it with #$00 when a bullet is not on the screen and with #$01 when one is. Thus testing BULON for #$00 or #$01 will tell us the bullet status.

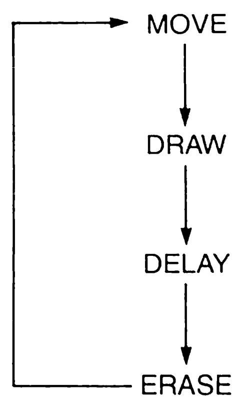
The bullet shape itself is just a single dot, both for convenience and also because it looks fine that way. This simplifies the BTEMP (B for Bullet) loading and also the draw routine. For example, a bullet shape is loaded by retrieving the shape address and loading its contents directly into BTEMP without the need for any counters (see the LOADBUL subroutine). Also, because the bullet shape has a depth of one, the draw routine has no need for XCOUNT, BLINEA, or CMP DEPTH; drawing (and erasing) is accomplished simply by LDA (LOW),Y, EOR BTEMP, STA (LOW),Y (see the BDRAW subroutine). In addition, because the first bullet shape occupies only the first or leftmost bit, the BSHAPE table need only consist of one screen byte—there is no need for an extra byte in the direction of movement.

The bullet is drawn moving up eight lines at a time. The reason for this is not apparent in Program 6-3 but will be when we incorporate the routine into the final game program. The reason is speed. In the game program, we want the bullet to move much faster than a plane moving across the screen. The plane and bullet move once per drawing cycle and the plane moves in 1-bit increments. Moving the bullet up one line at a time, for example, would slow it down so much relative to the plane as to detract seriously from whatever simulation of realism we hope to achieve. Although eight line moves may not be appropriate for most shapes, it works fine with a single dot and the animation simulates a fired bullet quite well.

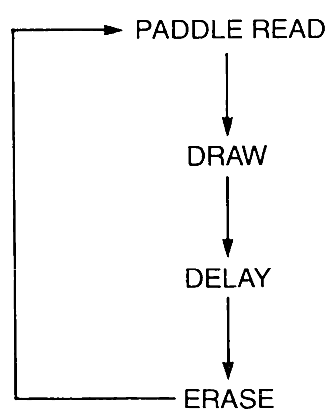
Finally, we have to test for the bullet reaching the top of the screen, at which point it is erased in preparation for the next firing. The bullet starts its screen traversal at line #$A4, just above the raised arm of the man. If we keep subtracting 8 from this value to get to a line position near the top of the screen, the nearest line to the top turns out to be 4. Thus, we do a CMP #$05 (line 112); if the line position is less than this, we've reached the top; if more, we continue drawing the bullet.

So much for the easy part, now for the mind-bender. Designing a complicated program, such as one that integrates multiple moving shapes, is best done, at least in my experience, by paying particular attention to the program flowchart, long before getting into extensive program details (this is always a good idea but is especially important for difficult programs). We'll be examining the flowchart for Program 6-4 in detail shortly. A further point regarding multiple shape programs, and one alluded to briefly in the bullet section above, is that each shape has essentially its own program within a program. For example, in Program 6-4, the bullet shape employs BINITIAL, BDRAW, BSHPADR, BHORIZ, etc., while the person shape uses its own set of labels and routines such as MINITIAL, MDRAW, MHORIZ, etc. (M stands for Man; let's face it, a man is more likely to be firing bullets at passing planes than a woman [a sad commentary?]. In any case, P for Person is not used because it is used for Plane in later programs.) The use of these separate routines and labels is a necessity, but a welcome one, because they make the program much easier to write and read.

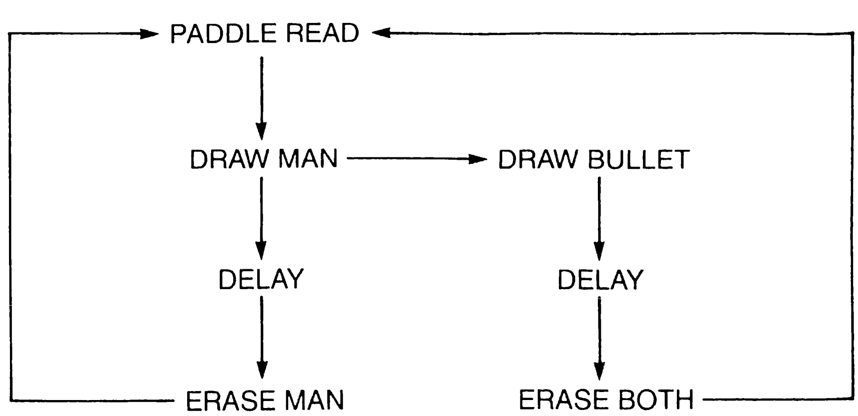
Remember the principles of animation discussed in previous chapters; i.e.:



We can modify this for paddle control as follows:

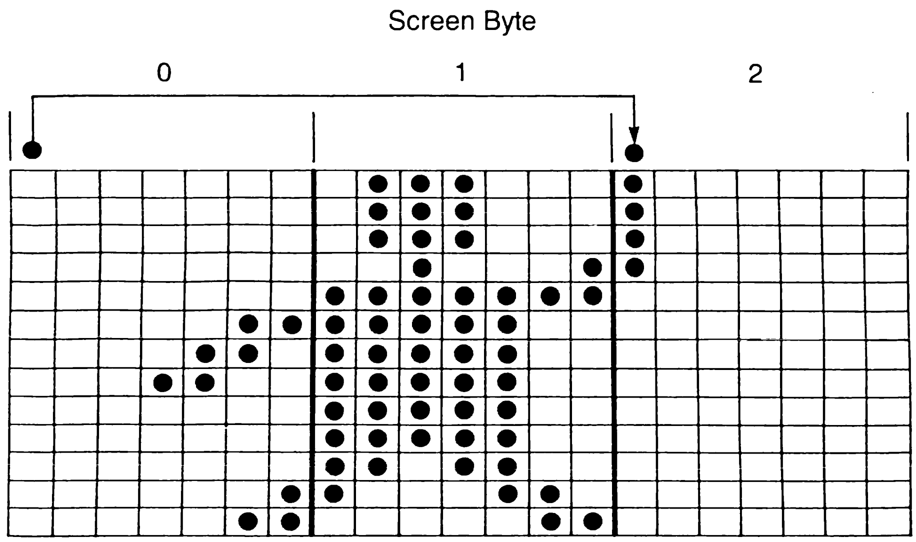


When drawing multiple shapes whose movement is controlled by a paddle, the diagram looks like this (without paddle control, substitute Move for Paddle):



Let's now examine in detail the flowchart for Program 6-4.

Courage, we're almost done. Because we've seen all the routines before, there is no need to discuss the details of Program 6-4 except for one point. The following diagram illustrates the position of the man shape when screen byte 0 and shape 0 are selected by the paddle read.



Assuming the bullet is ready to be fired, i.e., it is not already on the screen, the same paddle read also selects screen byte 0 and shape 0 for the bullet shape. B shape 0 is a single dot drawn at the leftmost bit position (shape byte #$01) and, if drawn in screen byte 0, will appear to be fired from a position remote from the upraised arm that holds the gun. To align the bullet exactly with the upraised arm, in the LOADBUL subroutine we add 2 to the bullet screen byte position (lines 192 and 193) before drawing. Thus, in this example, bullet shape 0 will be drawn in the leftmost bit position of screen byte 2 and will appear to emerge from the proper position. This relationship holds true throughout the screen range regardless of screen byte or shape number. This is the reason the man shape is drawn the way it is—remember the discussion in Chapter 5 regarding positioning the upraised arm in the leftmost bit position of the third screen byte of the shape table.

```
]PROGRAM 6-4
:ASM
                  1     *PADDLE CONTROL OF HORIZONTAL MOVEMENT AND SHOOTING BULLETS
                  2               ORG   $6000
6000: 4C 52 60    3               JMP   PGM
                  4     MLINE     DS    1
                  5     MLINEA    DS    1
                  6     BLINE     DS    1
                  7     DEPTH     DS    1
                  8     MHORIZ    DS    1
                  9     BHORIZ    DS    1
                  10    HORIZB    DS    1
                  11    HORIZM    DS    1
                  12    BULON     DS    1
                  13    XCOUNT    DS    1
                  14    DELAY     DS    1
                  15    BTEMP     DS    1
                  16    MTEMP     DS    39
```

```
                GRAPHICS =     $C050
                MIXOFF   =     $C052
                HIRES    =     $C057
                PAGE1    =     $C054
                HIGH     =     $1B
                LOW      =     $1A
                WAIT     =     $FCA8
                PREAD    =     $FB1E
                BUTTON   =     $C061      ;BUTTON 0
                *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                *CONTINUE FOR ALL 7 SHAPES
6036: BE        MSHPADR  DFB   #<MSHAPE1
6037: 61                 DFB   #>MSHAPE1
6038: E5                 DFB   #<MSHAPE2
6039: 61                 DFB   #>MSHAPE2
603A: 0C                 DFB   #<MSHAPE3
603B: 62                 DFB   #>MSHAPE3
603C: 33                 DFB   #<MSHAPE4
603D: 62                 DFB   #>MSHAPE4
603E: 5A                 DFB   #<MSHAPE5
603F: 62                 DFB   #>MSHAPE5
6040: 81                 DFB   #<MSHAPE6
6041: 62                 DFB   #>MSHAPE6
6042: A8                 DFB   #<MSHAPE7
6043: 62                 DFB   #>MSHAPE7
6044: CF        BSHPADR  DFB   #<BSHAPE1
6045: 62                 DFB   #>BSHAPE1
6046: D0                 DFB   #<BSHAPE2
6047: 62                 DFB   #>BSHAPE2
6048: D1                 DFB   #<BSHAPE3
6049: 62                 DFB   #>BSHAPE3
604A: D2                 DFB   #<BSHAPE4
604B: 62                 DFB   #>BSHAPE4
604C: D3                 DFB   #<BSHAPE5
604D: 62                 DFB   #>BSHAPE5
604E: D4                 DFB   #<BSHAPE6
604F: 62                 DFB   #>BSHAPE6
6050: D5                 DFB   #<BSHAPE7
6051: 62                 DFB   #>BSHAPE7
6052: AD 50 C0  PGM      LDA   GRAPHICS   ;HIRES,P.1
6055: AD 52 C0           LDA   MIXOFF
6058: AD 57 C0           LDA   HIRES
605B: AD 54 C0           LDA   PAGE1
605E: A9 00              LDA   #$00       ;CLEAR SCREEN 1
6060: 85 1A              STA   LOW
6062: A9 20              LDA   #$20
6064: 85 1B              STA   HIGH
6066: A0 00     CLR1     LDY   #$00
6068: A9 00              LDA   #$00
606A: 91 1A     CLR      STA   (LOW),Y
606C: C8                 INY
606D: D0 FB              BNE   CLR
606F: E6 1B              INC   HIGH
6071: A5 1B              LDA   HIGH
6073: C9 40              CMP   #$40
6075: 90 EF              BLT   CLR1
6077: A9 50              LDA   #$50       ;LOAD DELAY
6079: 8D 0D 60           STA   DELAY
607C: A2 B7              LDX   #$B7       ;DRAW LINE
607E: A0 00              LDY   #$00
6080: BD DC 64           LDA   HI,X
```

```
6199: BD 45 60  200              LDA  BSHPADR+1,X
619C: 85 1B     201              STA  HIGH
619E: A0 00     202              LDY  #$00
61A0: B1 1A     203              LDA  (LOW),Y
61A2: 8D 0E 60  204              STA  BTEMP
61A5: 60        205              RTS
                206    **********************
61A6: AE 05 60  207    BDRAW     LDX  BLINE
61A9: AC 09 60  208              LDY  HORIZB
61AC: BD DC 64  209              LDA  HI,X
61AF: 85 1B     210              STA  HIGH
61B1: BD 9C 65  211              LDA  LO,X
61B4: 85 1A     212              STA  LOW
61B6: B1 1A     213              LDA  (LOW),Y
61B8: 4D 0E 60  214              EOR  BTEMP
61BB: 91 1A     215              STA  (LOW),Y
61BD: 60        216              RTS
61BE: 00 0E 01  217    MSHAPE1   HEX  000E01000E01000E01     ;MAN SHAPE TABLES
61C1: 00 0E 01 00 0E 01
61C7: 00 44 01  218              HEX  004401007F00601F00
61CA: 00 7F 00 60 1F 00
61D0: 30 1F 00  219              HEX  301F00181F00001F00
61D3: 18 1F 00 00 1F 00
61D9: 00 1F 00  220              HEX  001F00001B00403100
61DC: 00 1B 00 40 31 00
61E2: 60 60 00  221              HEX  606000
61E5: 00 1C 02  222    MSHAPE2   HEX  001C02001C02001C02
61E8: 00 1C 02 00 1C 02
61EE: 00 08 03  223              HEX  000803007E01003E00
61F1: 00 7E 01 00 3E 00
61F7: 00 3F 00  224              HEX  003F00403F00003E00
61FA: 40 3F 00 00 3E 00
6200: 00 3E 00  225              HEX  003E00003600003600
6203: 00 36 00 00 36 00
6209: 00 63 00  226              HEX  006300
620C: 00 38 04  227    MSHAPE3   HEX  003804003804003804
620F: 00 38 04 00 38 04
6215: 00 10 06  228              HEX  001006007C03007C00
6218: 00 7C 03 00 7C 00
621E: 00 7C 00  229              HEX  007C00007E00007C00
6221: 00 7E 00 00 7C 00
6227: 00 38 00  230              HEX  003800003800006C00
622A: 00 38 00 00 6C 00
6230: 00 46 01  231              HEX  004601
6233: 00 70 08  232    MSHAPE4   HEX  007008007008007008
6236: 00 70 08 00 70 08
623C: 00 20 0C  233              HEX  00200C007807007801
623F: 00 78 07 00 78 01
6245: 00 78 01  234              HEX  007801007801007801
6248: 00 78 01 00 78 01
624E: 00 70 00  235              HEX  007000007000007000
6251: 00 70 00 00 70 00
6257: 00 70 00  236              HEX  007000
625A: 00 60 11  237    MSHAPE5   HEX  006011006011006011
625D: 00 60 11 00 60 11
6263: 00 40 18  238              HEX  00401800700F007003
6266: 00 70 0F 00 70 03
626C: 00 70 03  239              HEX  007003007803007003
626F: 00 78 03 00 70 03
6275: 00 60 01  240              HEX  006001006001003003
6278: 00 60 01 00 30 03
```

```
627E: 00 18 06  241           HEX  001806
6281: 00 40 23  242  MSHAPE6  HEX  004023004023004023
6284: 00 40 23 00 40 23
628A: 00 00 31  243           HEX  00003100601F006007
628D: 00 60 1F 00 60 07
6293: 00 70 07  244           HEX  007007007807006007
6296: 00 78 07 00 60 07
629C: 00 60 07  245           HEX  006007006006006006
629F: 00 60 06 00 60 06
62A5: 00 30 0C  246           HEX  00300C
62A8: 00 00 47  247  MSHAPE7  HEX  000047000047000047
62AB: 00 00 47 00 00 47
62B1: 00 00 62  248           HEX  00006200403F00700F
62B4: 00 40 3F 00 70 0F
62BA: 00 58 0F  249           HEX  00580F004C0F00400F
62BD: 00 4C 0F 00 40 0F
62C3: 00 40 0F  250           HEX  00400F00400D006018
62C6: 00 40 0D 00 60 18
62CC: 00 30 30  251           HEX  003030
62CF: 01        252  BSHAPE1  HEX  01              ;BULLET SHAPES
62D0: 02        253  BSHAPE2  HEX  02
62D1: 04        254  BSHAPE3  HEX  04
62D2: 08        255  BSHAPE4  HEX  08
62D3: 10        256  BSHAPE5  HEX  10
62D4: 20        257  BSHAPE6  HEX  20
62D5: 40        258  BSHAPE7  HEX  40
                     BYTETBL
                     OFFSET
                     HI
                     LO

1628 bytes

Symbol table - numerical order:

   LOW      =$1A      HIGH     =$1B      MLINE    =$6003    MLINEA   =$6004
   BLINE    =$6005    DEPTH    =$6006    MHORIZ   =$6007    BHORIZ   =$6008
   HORIZB   =$6009    HORIZM   =$600A    BULON    =$600B    XCOUNT   =$600C
   DELAY    =$600D    BTEMP    =$600E    MTEMP    =$600F    MSHPADR  =$6036
   BSHPADR  =$6044    PGM      =$6052    CLR1     =$6066    CLR      =$606A
   LN       =$608C    PADDLE   =$6099    BULLET1  =$60B7    BULLET   =$60BC
   TOP      =$60DE    MINITIAL=$60E4     BINITIAL=$60F3     PDLE     =$60FE
   PDLE1    =$6114    LOAD     =$612E    MDRAW    =$6139    MDRAW1   =$613E
   LOADBUL  =$6183    BDRAW    =$61A6    MSHAPE1  =$61BE    MSHAPE2  =$61E5
   MSHAPE3  =$620C    MSHAPE4  =$6233    MSHAPE5  =$625A    MSHAPE6  =$6281
   MSHAPE7  =$62A8    BSHAPE1  =$62CF    BSHAPE2  =$62D0    BSHAPE3  =$62D1
   BSHAPE4  =$62D2    BSHAPE5  =$62D3    BSHAPE6  =$62D4    BSHAPE7  =$62D5
   BYTETBL  =$62D6    OFFSET   =$63D9    HI       =$64DC    LO       =$659C
   GRAPHICS=$C050     MIXOFF   =$C052    PAGE1    =$C054    HIRES    =$C057
   BUTTON   =$C061    PREAD    =$FB1E    WAIT     =$FCA8
```

We've now gotten through the most difficult part of our discussion of game design. The next few chapters will discuss collisions and explosions, scoring, sound, and in Chapter 10, assembling the final game which involves more or less the same technique developed in this chapter, i.e., a detailed examination of the flowchart, except on a larger scale.

7

Collisions and Explosions

A scientist from the war games division
Designed a game with the ultimate collision.
Out of mutual fear
Missiles went flying in air,
And then, a final nuclear fission.

Collision detection is an integral part of almost every game program and can be used for almost any purpose because once a collision is detected, the program can be instructed to do a multitude of things. For example, a shape can be constricted to the lanes of a maze by not allowing movement past lane boundaries if a collision with these boundaries is detected. In our game program, we're going to detect a collision of a bullet with a passing plane and this will be followed by drawing explosion shapes at the area of impact. In later chapters we will see how to integrate sound and scoring with these collision events.

COLLISION DETECTION

The heart of collision detection is the AND instruction. AND compares each bit of the Accumulator with the corresponding bit of a byte, either a direct value or the contents of a memory location, and returns a value of 1 if both bits are 1; otherwise, the result will be 0. The result is stored in the Accumulator.

Example

```
Accumulator               0 0 1 1 0 0 1 1
AND byte                  0 1 0 1 0 1 0 1
                          ---------------
Result in Accumulator     0 0 0 1 0 0 0 1
```

Let's see how we can use this instruction to detect collisions. Suppose we have a bullet shape, a single dot, moving up the screen. We want to ask: does the corresponding bit position of the next screen byte position the bullet is going to move into contain a 1 (i.e., a shape) or a 0 (i.e., no shape)? If the screen bit contains a 0, ANDing the screen byte with the bullet shape byte will return a 0; if it contains a 1, ANDing will return some number greater than 0.

Example

No collision		Shape on Screen
Screen byte in Accumulator (#$7E)	0 1 1 1 1 1 1 0	0 1 1 1 1 1 1
AND with bullet shape (#$01)	0 0 0 0 0 0 0 1	1 0 0 0 0 0 0
Result in Accumulator (zero)	0 0 0 0 0 0 0 0	
Collision		Shape on Screen
Screen byte in Accumulator (#$7E)	0 1 1 1 1 1 1 0	0 1 1 1 1 1 1
AND with bullet shape (#$02)	0 0 0 0 0 0 1 0	0 1 0 0 0 0 0
Result in Accumulator (non-zero)	0 0 0 0 0 0 1 0	

Note that the screen byte itself does not have to be entirely empty for there to be no collision; only the corresponding bit position must be empty. This is exactly what we want. Suppose the screen byte #$7E corresponds to the bottom line of a plane shape moving left to right. If a bullet is to move into this screen byte and if the bullet shape byte, by virtue of its horizontal positioning, is #$01, the AND instruction will return a value of 0, i.e., no collision, which describes the situation perfectly; the bullet will pass just to the left of the plane (see Shape on Screen column above). On the other hand, if the bullet shape byte, by virtue of its horizontal positioning, is #$02, the bullet will appear to hit the plane and the AND instruction will return a value greater than 0, i.e., a collision. (There is an obvious problem here if the high or leftmost bit is set to 1 for either the screen or shape byte as it is for selecting some colors but we'll get to this problem in the chapter on drawing in color—for now, and for all the programs in Part One, the high bit is set to 0).

Now that these principles have collided with your brain cells, we can describe a general routine for collision detection as follows:

```
LDA Screen Byte
AND Shape Byte      ;RESULT IS ZERO IF NO COLLISION
CMP #$00
BEQ NOHIT           ;BRANCH TO NOHIT IF NO COLLISION
JMP COLLISION       ;GO TO COLLISION IF COLLISION
```

The CMP #$00 is not really needed here, as BEQ will branch when the result of the previous operation is zero, but it is included to make the program easier to read.

This general routine presents a problem when we want to detect a collision with shapes moving non-vertically using a DRAW-DRAW protocol. Vertical movement with DRAW-DRAW is okay—the screen byte to be AND'ed is one or more lines above or below the shape and is either empty or not. However, non-vertical movement always contains a horizontal vector and in horizontal movement, the same screen byte is repeatedly accessed for each of the seven shapes. Thus, if we use DRAW-DRAW and the AND test for horizontal movement, the first time we draw a shape we're okay. But when we want to draw the next shape, the same screen byte is accessed (except at the screen byte boundaries) and AND'ed with the shape byte. The screen byte still contains the first shape byte because there is no erase cycle and thus a collision will be detected. In other words, the shape will continually "collide" with itself. Therefore if a shape with horizontal movement is itself to be used for collision detection, it must use the DRAW-ERASE protocol. DRAW-ERASE works because the shape byte is erased before the collision test. Note that with DRAW-ERASE, the shape byte and not the screen byte is erased. Thus, if the screen byte contains an "on" bit from another shape, this bit will not be erased by EOR Shape Byte because with EOR, 0 + 1 = 1. Thus:

		Shape 1	Shape 2
LDA Screen Byte	1 1 1 0 0 0 1	#$07	#$40
EOR Shape 1 Byte	1 1 1 0 0 0 0		
Result in Accumulator	0 0 0 0 0 0 1	Shape 2 still in screen byte	

As the first shape is both drawn and erased with EOR, the bit from the second shape is always present for the AND test and a collision will result when the first shape enters this bit position.

Let's put all this to work in an actual program. The next program (Program 7-1) is essentially the same as Program 6-3 except now we've drawn a line in the left half of the screen near the top—if a fired bullet hits this line, a long delay will ensue as a collision marker. Shooting the bullet in the right half of the screen will, of course, result in no collision, as there's no line there.

Now to the details of Program 7-1. First, we draw a line in the left half of the screen at screen line position #$0C (12). There's a good reason for drawing the line at that particular line position as we'll soon see. Next, the program continues in the same way as Program 6-3 until we get to the point where the answer to the questions, "Is the Bullet On?" or "Is the Button Pressed?" is yes.

Now, instead of just drawing the bullet, we first test for a collision; i.e., is there something in the screen bit position where the bullet is to be drawn? The BDRAW routine specifies the line and screen byte where the bullet is to be drawn from BLINE and HORIZB. The instruction LDA (LOW),Y (line 224) loads the Accumulator with the screen byte contents and the next instruction, AND BTEMP, AND's the Accumulator with the bullet shape byte (remember that the value in BTEMP is determined by the horizontal position of the man when the bullet is fired). If the AND result is zero, there will be no collision, the program branches to NOHIT where the bullet is drawn, and the program continues just as in Program 6-3. If the AND result is non-zero, this indicates a collision and the program jumps to COLLISION, which produces a long delay simply as a collision marker, and then initializes the bullet, erases the man, and jumps back for another paddle read. (The BPL instruction [line 238] in the long-delay loop in the COLLISION subroutine continues the delay loop until Y = #$FF; because Y initially contains #$10, the LDA #$FF, JSR WAIT delay will loop 17 times before going to JSR BINITIAL.)

There are a few other details of Program 7-1 we have to consider before going on. First, you might have noticed from the flowchart that the bullet seems not to have been erased after the collision. In fact, it has, because the collision test occurs before the bullet is drawn, not after. In other words, the sequence is draw-erase-test, draw-erase-test, etc. This seems to present another problem, because the bullet is moved up eight lines at a time and thus the last bullet on the screen is eight lines below the collision site. In actuality, however, the bullet is moving so fast that the illusion of a direct hit is preserved. In any event, this is a special situation that arises only when a shape to be tested for a collision is moved large distances between each test. With the more usual smaller moves, say one or two lines or bits at a time, the direct hit illusion is preserved even with slow-moving shapes—the eye can hardly discern whether a collision is on target or one or two bits or lines away. In any case, if this bothers you, you could incorporate the following routine, which uses a test-draw-erase cycle. The shape is drawn in its next position whether or not a collision has been detected—if detected, COLL is set to 1 and this branches the program to COLLISION, which erases the shape before continuing:

```
           LDA  (LOW),Y
           AND  BTEMP
           CMP  #$00
           BEQ  NOHIT
           LDA  #$01       ;LOAD COLL WITH ONE IF COLLISION
           STA  COLL
NOHIT      LDA  (LOW),Y    ;DRAW BULLET
           EOR  BTEMP
           STA  (LOW),Y
           LDA  COLL
           CMP  #$01
           BEQ  COLLISION  ;JUMP TO COLLISION IF COLLISION
           RTS             ;ELSE RETURN TO MAIN PROGRAM
COLLISION  JSR  BXDRAW     ;ERASE BULLET
           etc.
```


Next, you will notice that when the bullet is erased after no collision, we access a routine called BXDRAW instead of BDRAW. This is because BDRAW contains the collision test instructions. If we access BDRAW for the bullet erase, LDA (LOW),Y would load the Accumulator with the content of the screen byte, which is in fact the bullet shape byte because the bullet is already on the screen at that location. Thus, if we then do an AND BTEMP, a collision will always be detected even though the bullet isn't hitting anything (except itself!). Therefore we use BXDRAW to erase—BXDRAW is the same as BDRAW but without the collision test instructions.

Finally, we have to discuss how to ensure that the shapes to collide will occupy the same bit positions at the apparent point of collision, a not inconsequential problem. If a shape to be tested for a collision is moved one bit or line at a time, there is no problem, but if the shape moves in larger increments, the collision test may fail even though a collision appears to take place on the screen. For example, the bullet shape in Program 7-1 is tested for a collision at only every eighth line (because it moves up eight lines at a time) starting from line 164 (#$A4). Thus, a collision will be detected only with shapes that occupy a screen line some multiple of 8 from the starting line—this is why the top line is drawn at line 12 (#$0C).

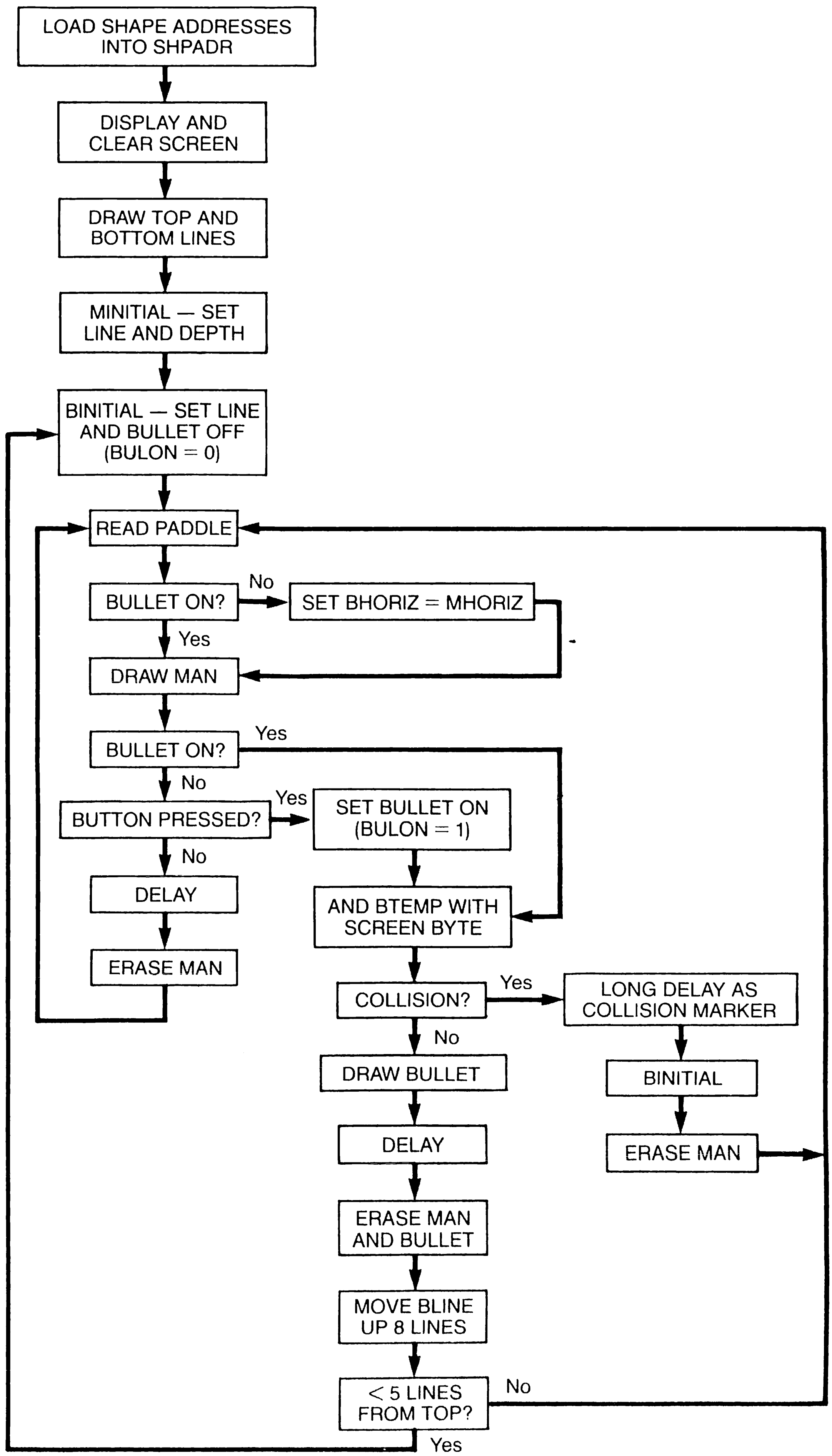
Try this for yourself. Draw the top line at screen line 11 or 13 and run the program—the bullet will appear to go right through the line with no collision. However, this appears to be much more of a problem than it is. First, in most cases, shapes are moved only one line or bit at a time and in this situation, every screen line or bit position will be collision-tested. Second, in the case of larger movements, such as the bullet move, all we need do is ensure that the shape to be collided with is in the proper position. In the final game program, for example, bullets are fired at passing planes and all we have to do is draw the planes or some part of the planes at screen lines some multiple of 8 from the bullet starting line. Remember, we are now expert assembly language programmers and so we can draw shapes anywhere we want!

Suppose, however, we can't predict the screen position of a shape to be collided with. For example, suppose we modify the game program so that the planes drop bombs and we want to detect collisions of the bullet with the bombs as well as the planes. The bombs drop in a parabolic curve and at the point of apparent collision with the bullet, may or may not be at one of the multiple of 8 line positions. To get around this, we can use the following BDRAW routine which collision-tests every line position from the last bullet drawn, not just the eighth position up:

```
BDRAW   LDA BLINE       ;BL IS LOADED WITH BLINE AND
        STA BL          ;CTR WITH BLINE + UP 7 LINES
        SEC
        SBC #$07
        STA CTR
************************ TEST COLLISION FROM BL TO CTR ************************
COL     LDX BL          ;TEST COLLISION FOR LINE BL
        LDY HORIZB
        LDA HI,X
        STA HIGH
        LDA LO,X
        STA LOW
        LDA (LOW),Y
        AND BTEMP
        CMP #$00        ;IF NO COLLISION, GO TO COL1 TO TEST NEXT LINE
        BEQ COL1
        JMP COLLISION   ;IF COLLISION, JUMP TO COLLISION
COL1    DEC BL          ;TEST NEXT LINE UP
        LDA BL
        CMP CTR
        BGE COL         ;STOP TEST WHEN LINE REACHES CTR
        LDX BLINE       ;IF NO COLLISION, DRAW BULLET
        LDY HORIZB
        LDA HI,X
        STA HIGH
        LDA LO,X
        STA LOW
        LDA (LOW),Y
        EOR BTEMP
        STA (LOW),Y
        RTS
```


Try this routine in Program 7-1. You will find that the bullet will collide with any shape regardless of its screen line position.

One final word about shape positions and collisions. If the shape to be collided with is larger than the movement of the collision test shape, the above type of routine would not be necessary. For example, if we want to test for the collision of a bullet with a shape that at every point is at least 8 lines deep, then obviously some part of the shape will always be at a line position that is some multiple of 8 from the bullet starting line.


LOAD SHAPE ADDRESSES INTO SHPADR
DISPLAY AND CLEAR SCREEN
DRAW TOP AND BOTTOM LINES
MINITIAL — SET LINE AND DEPTH
BINITIAL — SET LINE AND BULLET OFF (BULON = 0)
READ PADDLE
BULLET ON?
No
SET BHORIZ = MHORIZ
Yes
DRAW MAN
BULLET ON?
Yes
No
BUTTON PRESSED?
Yes
SET BULLET ON (BULON = 1)
No
DELAY
AND BTEMP WITH SCREEN BYTE
ERASE MAN
COLLISION?
Yes
LONG DELAY AS COLLISION MARKER
No
DRAW BULLET
BINITIAL
DELAY
ERASE MAN
ERASE MAN AND BULLET
MOVE BLINE UP 8 LINES
< 5 LINES FROM TOP?
No
Yes

```
]PROGRAM 7-1
:ASM
                *TESTING FOR COLLISION*
                         ORG   $6000
6000: 4C 52 60           JMP   PGM
                MLINE    DS    1
                MLINEA   DS    1
                BLINE    DS    1
                DEPTH    DS    1
                MHORIZ   DS    1
                BHORIZ   DS    1
                HORIZB   DS    1
                HORIZM   DS    1
                BULON    DS    1
                XCOUNT   DS    1
                DELAY    DS    1
                BTEMP    DS    1
                MTEMP    DS    39
                GRAPHICS =     $C050
                MIXOFF   =     $C052
                HIRES    =     $C057
                PAGE1    =     $C054
                HIGH     =     $1B
                LOW      =     $1A
                WAIT     =     $FCA8
                PREAD    =     $FB1E
                BUTTON   =     $C061      ;BUTTON 0
                *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                *CONTINUE FOR ALL 7 SHAPES
6036: 0C        MSHPADR  DFB   #<MSHAPE1
6037: 62                 DFB   #>MSHAPE1
6038: 33                 DFB   #<MSHAPE2
6039: 62                 DFB   #>MSHAPE2
603A: 5A                 DFB   #<MSHAPE3
603B: 62                 DFB   #>MSHAPE3
603C: 81                 DFB   #<MSHAPE4
603D: 62                 DFB   #>MSHAPE4
603E: A8                 DFB   #<MSHAPE5
603F: 62                 DFB   #>MSHAPE5
6040: CF                 DFB   #<MSHAPE6
6041: 62                 DFB   #>MSHAPE6
6042: F6                 DFB   #<MSHAPE7
6043: 62                 DFB   #>MSHAPE7
6044: 1D        BSHPADR  DFB   #<BSHAPE1
6045: 63                 DFB   #>BSHAPE1
6046: 1E                 DFB   #<BSHAPE2
6047: 63                 DFB   #>BSHAPE2
6048: 1F                 DFB   #<BSHAPE3
6049: 63                 DFB   #>BSHAPE3
604A: 20                 DFB   #<BSHAPE4
604B: 63                 DFB   #>BSHAPE4
604C: 21                 DFB   #<BSHAPE5
604D: 63                 DFB   #>BSHAPE5
604E: 22                 DFB   #<BSHAPE6
604F: 63                 DFB   #>BSHAPE6
6050: 23                 DFB   #<BSHAPE7
6051: 63                 DFB   #>BSHAPE7
6052: AD 50 C0  PGM      LDA   GRAPHICS   ;HIRES,P.1
6055: AD 52 C0           LDA   MIXOFF
6058: AD 57 C0           LDA   HIRES
605B: AD 54 C0           LDA   PAGE1
```

```
605E: A9 00                 LDA  #$00        ;CLEAR SCREEN 1
6060: 85 1A                 STA  LOW
6062: A9 20                 LDA  #$20
6064: 85 1B                 STA  HIGH
6066: A0 00        CLR1     LDY  #$00
6068: A9 00                 LDA  #$00
606A: 91 1A        CLR      STA  (LOW),Y
606C: C8                    INY
606D: D0 FB                 BNE  CLR
606F: E6 1B                 INC  HIGH
6071: A5 1B                 LDA  HIGH
6073: C9 40                 CMP  #$40
6075: 90 EF                 BLT  CLR1
6077: A9 50                 LDA  #$50        ;LOAD DELAY
6079: 8D 0D 60              STA  DELAY
607C: A2 B7                 LDX  #$B7        ;DRAW BOTTOM LINE
607E: A0 00                 LDY  #$00
6080: BD 2A 65              LDA  HI,X
6083: 85 1B                 STA  HIGH
6085: BD EA 65              LDA  LO,X
6088: 85 1A                 STA  LOW
608A: A9 7F                 LDA  #$7F
608C: 91 1A        LN       STA  (LOW),Y
608E: C8                    INY
608F: C0 27                 CPY  #$27
6091: 90 F9                 BLT  LN
6093: A2 0C                 LDX  #$0C        ;DRAW TOP LINE
6095: A0 00                 LDY  #$00
6097: BD 2A 65              LDA  HI,X
609A: 85 1B                 STA  HIGH
609C: BD EA 65              LDA  LO,X
609F: 85 1A                 STA  LOW
60A1: A9 7F                 LDA  #$7F
60A3: 91 1A        LN1      STA  (LOW),Y
60A5: C8                    INY
60A6: C0 14                 CPY  #$14
60A8: 90 F9                 BLT  LN1
                   ********** MAIN PROGRAM **********
60AA: 20 FB 60              JSR  MINITIAL    ;SET LINE & DEPTH OF MAN
60AD: 20 0A 61              JSR  BINITIAL    ;SET LINE FOR BULLET
60B0: 20 15 61     PADDLE   JSR  PDLE        ;READ PADDLE
60B3: 20 50 61              JSR  MDRAW       ;DRAW MAN
60B6: AD 0B 60              LDA  BULON
60B9: C9 01                 CMP  #$01        ;IS BULLET ON?
60BB: F0 16                 BEQ  BULLET      ;IF YES, CONTINUE BULLET DRAW
60BD: AD 61 C0              LDA  BUTTON      ;IF NO, IS BUTTON PRESSED?
60C0: 30 0C                 BMI  BULLET1     ;IF YES, DRAW BULLET
60C2: AD 0D 60              LDA  DELAY       ;IF NO,
60C5: 20 A8 FC              JSR  WAIT            DELAY AND
60C8: 20 50 61              JSR  MDRAW           ERASE MAN AND
60CB: 4C B0 60              JMP  PADDLE          READ PADDLE AGAIN
60CE: A9 01        BULLET1  LDA  #$01        ;SET BULLET ON
60D0: 8D 0B 60              STA  BULON
60D3: 20 9A 61     BULLET   JSR  LOADBUL     ;LOAD BULLET SHAPE INTO BTEMP
60D6: 20 BD 61              JSR  BDRAW       ;DRAW BULLET & TEST FOR COLLISION
60D9: AD 0D 60              LDA  DELAY
60DC: 20 A8 FC              JSR  WAIT        ;DELAY
60DF: 20 F4 61              JSR  BXDRAW      ;ERASE BULLET
60E2: 20 50 61              JSR  MDRAW       ;ERASE MAN
60E5: AD 05 60              LDA  BLINE
60E8: 38                    SEC
```

```
60E9: E9 08              SBC  #$08          ;MOVE BLINE UP 8 LINES
60EB: 8D 05 60           STA  BLINE
60EE: C9 05              CMP  #$05          ;LESS THAN 5 LINES FROM TOP?
60F0: 90 03              BLT  TOP           ;IF YES TAKE BRANCH
60F2: 4C B0 60           JMP  PADDLE        ;IF NO, READ PADDLE AGAIN
60F5: 20 0A 61     TOP   JSR  BINITIAL      ;INITIALIZE BULLET LINE
60F8: 4C B0 60           JMP  PADDLE        ;READ PADDLE
                   ********** SUBROUTINES **********
60FB: A9 AA        MINITIAL LDA #$AA
60FD: 8D 03 60           STA  MLINE
6100: 8D 04 60           STA  MLINEA
6103: 18                 CLC
6104: 69 0D              ADC  #$0D
6106: 8D 06 60           STA  DEPTH
6109: 60                 RTS
                   ***********************
610A: A9 00        BINITIAL LDA #$00        ;BULON = 0 IF
610C: 8D 0B 60           STA  BULON             BULLET NOT ON SCREEN
610F: A9 A4              LDA  #$A4
6111: 8D 05 60           STA  BLINE
6114: 60                 RTS
                   **********************
6115: A2 00        PDLE  LDX  #$00
6117: 20 1E FB           JSR  PREAD         ;READ PADDLE 0
611A: 98                 TYA
611B: 8D 07 60           STA  MHORIZ        ;0-255 IN MHORIZ
611E: AD 0B 60           LDA  BULON
6121: C9 01              CMP  #$01          ;IS BULLET ON?
6123: F0 06              BEQ  PDLE1         ;IF YES, TAKE BRANCH
6125: AD 07 60           LDA  MHORIZ        ;IF NO, SET BHORIZ EQUAL
6128: 8D 08 60           STA  BHORIZ            TO MHORIZ
612B: AC 07 60     PDLE1 LDY  MHORIZ
612E: B9 24 63           LDA  BYTETBL,Y     ;CONVERT 0-255 TO 0-36 (BYTE)
6131: 8D 0A 60           STA  HORIZM        ;MAN BYTE POSITION
6134: B9 27 64           LDA  OFFSET,Y      ;GET SHAPE NUMBER
6137: 0A                 ASL                ;LOAD SHAPE INTO MTEMP
6138: AA                 TAX
6139: BD 36 60           LDA  MSHPADR,X
613C: 85 1A              STA  LOW
613E: BD 37 60           LDA  MSHPADR+1,X
6141: 85 1B              STA  HIGH
6143: A0 00              LDY  #$00
6145: B1 1A        LOAD  LDA  (LOW),Y
6147: 99 0F 60           STA  MTEMP,Y
614A: C8                 INY
614B: C0 27              CPY  #$27
614D: 90 F6              BLT  LOAD
614F: 60                 RTS
                   ***********************
6150: A9 00        MDRAW LDA  #$00
6152: 8D 0C 60           STA  XCOUNT
6155: AE 03 60     MDRAW1 LDX MLINE
6158: AC 0A 60           LDY  HORIZM
615B: BD 2A 65           LDA  HI,X
615E: 85 1B              STA  HIGH
6160: BD EA 65           LDA  LO,X
6163: 85 1A              STA  LOW
6165: AE 0C 60           LDX  XCOUNT
6168: B1 1A              LDA  (LOW),Y
616A: 5D 0F 60           EOR  MTEMP,X
616D: 91 1A              STA  (LOW),Y
```

```
616F: C8                       INY
6170: B1 1A                    LDA   (LOW),Y
6172: 5D 10 60                 EOR   MTEMP+1,X
6175: 91 1A                    STA   (LOW),Y
6177: C8                       INY
6178: B1 1A                    LDA   (LOW),Y
617A: 5D 11 60                 EOR   MTEMP+2,X
617D: 91 1A                    STA   (LOW),Y
617F: EE 0C 60                 INC   XCOUNT
6182: EE 0C 60                 INC   XCOUNT
6185: EE 0C 60                 INC   XCOUNT
6188: EE 03 60                 INC   MLINE
618B: AD 03 60                 LDA   MLINE
618E: CD 06 60                 CMP   DEPTH
6191: 90 C2                    BLT   MDRAW1
6193: AD 04 60                 LDA   MLINEA        ;RESET LINE
6196: 8D 03 60                 STA   MLINE
6199: 60                       RTS
                     **********************
619A: AC 08 60       LOADBUL   LDY   BHORIZ        ;CONVERTS 0-255 TO
619D: B9 24 63                 LDA   BYTETBL,Y         SCREEN BYTE (0-36)
61A0: 18                       CLC                 ;ADD 2 TO ALIGN BULLET
61A1: 69 02                    ADC   #$02              WITH GUN
61A3: 8D 09 60                 STA   HORIZB        ;BULLET BYTE POSITION
61A6: B9 27 64                 LDA   OFFSET,Y      ;GET BULLET SHAPE NUMBER
61A9: 0A                       ASL                 ;LOAD BULLET SHAPE INTO BTEMP
61AA: AA                       TAX
61AB: BD 44 60                 LDA   BSHPADR,X
61AE: 85 1A                    STA   LOW
61B0: BD 45 60                 LDA   BSHPADR+1,X
61B3: 85 1B                    STA   HIGH
61B5: A0 00                    LDY   #$00
61B7: B1 1A                    LDA   (LOW),Y
61B9: 8D 0E 60                 STA   BTEMP
61BC: 60                       RTS
                     **********************
61BD: AE 05 60       BDRAW     LDX   BLINE
61C0: AC 09 60                 LDY   HORIZB
61C3: BD 2A 65                 LDA   HI,X
61C6: 85 1B                    STA   HIGH
61C8: BD EA 65                 LDA   LO,X
61CB: 85 1A                    STA   LOW
61CD: B1 1A                    LDA   (LOW),Y
61CF: 2D 0E 60                 AND   BTEMP         ;RESULT IS 0 IF NO COLLISION
61D2: C9 00                    CMP   #$00
61D4: F0 03                    BEQ   NOHIT
61D6: 4C E1 61                 JMP   COLLISION
61D9: B1 1A          NOHIT     LDA   (LOW),Y       ;CONTINUE BULLET DRAW
61DB: 4D 0E 60                 EOR   BTEMP
61DE: 91 1A                    STA   (LOW),Y
61E0: 60                       RTS
                     ***********************
61E1: A0 10          COLLISION LDY #$10            ;LONG TIME DELAY
61E3: A9 FF          COL1      LDA   #$FF
61E5: 20 A8 FC                 JSR   WAIT
61E8: 88                       DEY
61E9: 10 F8                    BPL   COL1
61EB: 20 0A 61                 JSR   BINITIAL
61EE: 20 50 61                 JSR   MDRAW         ;ERASE MAN
61F1: 4C B0 60                 JMP   PADDLE
                     **********************
```

```
61F4: AE 05 60  243  BXDRAW   LDX  BLINE        ;BDRAW WITHOUT COLLISION TEST
61F7: AC 09 60  244           LDY  HORIZB
61FA: BD 2A 65  245           LDA  HI,X
61FD: 85 1B     246           STA  HIGH
61FF: BD EA 65  247           LDA  LO,X
6202: 85 1A     248           STA  LOW
6204: B1 1A     249           LDA  (LOW),Y
6206: 4D 0E 60  250           EOR  BTEMP
6209: 91 1A     251           STA  (LOW),Y
620B: 60        252           RTS
                253  **************************
620C: 00 0E 01  254  MSHAPE1  HEX  000E01000E01000E01   ;MAN SHAPE TABLES
620F: 00 0E 01 00 0E 01
6215: 00 44 01  255           HEX  004401007F00601F00
6218: 00 7F 00 60 1F 00
621E: 30 1F 00  256           HEX  301F00181F00001F00
6221: 18 1F 00 00 1F 00
6227: 00 1F 00  257           HEX  001F00001B00403100
622A: 00 1B 00 40 31 00
6230: 60 60 00  258           HEX  606000
6233: 00 1C 02  259  MSHAPE2  HEX  001C02001C02001C02
6236: 00 1C 02 00 1C 02
623C: 00 08 03  260           HEX  000803007E01003E00
623F: 00 7E 01 00 3E 00
6245: 00 3F 00  261           HEX  003F00403F00003E00
6248: 40 3F 00 00 3E 00
624E: 00 3E 00  262           HEX  003E00003600003600
6251: 00 36 00 00 36 00
6257: 00 63 00  263           HEX  006300
625A: 00 38 04  264  MSHAPE3  HEX  003804003804003804
625D: 00 38 04 00 38 04
6263: 00 10 06  265           HEX  001006007C03007C00
6266: 00 7C 03 00 7C 00
626C: 00 7C 00  266           HEX  007C00007E00007C00
626F: 00 7E 00 00 7C 00
6275: 00 38 00  267           HEX  003800003800006C00
6278: 00 38 00 00 6C 00
627E: 00 46 01  268           HEX  004601
6281: 00 70 08  269  MSHAPE4  HEX  007008007008007008
6284: 00 70 08 00 70 08
628A: 00 20 0C  270           HEX  00200C007807007801
628D: 00 78 07 00 78 01
6293: 00 78 01  271           HEX  007801007801007801
6296: 00 78 01 00 78 01
629C: 00 70 00  272           HEX  007000007000007000
629F: 00 70 00 00 70 00
62A5: 00 70 00  273           HEX  007000
62A8: 00 60 11  274  MSHAPE5  HEX  006011006011006011
62AB: 00 60 11 00 60 11
62B1: 00 40 18  275           HEX  00401800700F007003
62B4: 00 70 0F 00 70 03
62BA: 00 70 03  276           HEX  007003007803007003
62BD: 00 78 03 00 70 03
62C3: 00 60 01  277           HEX  006001006001003003
62C6: 00 60 01 00 30 03
62CC: 00 18 06  278           HEX  001806
62CF: 00 40 23  279  MSHAPE6  HEX  004023004023004023
62D2: 00 40 23 00 40 23
62D8: 00 00 31  280           HEX  00003100601F006007
62DB: 00 60 1F 00 60 07
62E1: 00 70 07  281           HEX  007007007807006007
```

```
62E4: 00 78 07 00 60 07
62EA: 00 60 07   282          HEX  006007006006006006
62ED: 00 60 06 00 60 06
62F3: 00 30 0C   283          HEX  00300C
62F6: 00 00 47   284 MSHAPE7  HEX  000047000047000047
62F9: 00 00 47 00 00 47
62FF: 00 00 62   285          HEX  00006200403F00700F
6302: 00 40 3F 00 70 0F
6308: 00 58 0F   286          HEX  00580F004C0F00400F
630B: 00 4C 0F 00 40 0F
6311: 00 40 0F   287          HEX  00400F00400D006018
6314: 00 40 0D 00 60 18
631A: 00 30 30   288          HEX  003030
631D: 01         289 BSHAPE1  HEX  01          ;BULLET SHAPES
631E: 02         290 BSHAPE2  HEX  02
631F: 04         291 BSHAPE3  HEX  04
6320: 08         292 BSHAPE4  HEX  08
6321: 10         293 BSHAPE5  HEX  10
6322: 20         294 BSHAPE6  HEX  20
6323: 40         295 BSHAPE7  HEX  40
                     BYTETBL
                     OFFSET
                     HI
                     LO

1706 bytes

Symbol table - numerical order:

  LOW      =$1A      HIGH     =$1B      MLINE    =$6003    MLINEA   =$6004
  BLINE    =$6005    DEPTH    =$6006    MHORIZ   =$6007    BHORIZ   =$6008
  HORIZB   =$6009    HORIZM   =$600A    BULON    =$600B    XCOUNT   =$600C
  DELAY    =$600D    BTEMP    =$600E    MTEMP    =$600F    MSHPADR  =$6036
  BSHPADR  =$6044    PGM      =$6052    CLR1     =$6066    CLR      =$606A
  LN       =$608C    LN1      =$60A3    PADDLE   =$60B0    BULLET1  =$60CE
  BULLET   =$60D3    TOP      =$60F5    MINITIAL=$60FB     BINITIAL=$610A
  PDLE     =$6115    PDLE1    =$6123    LOAD     =$6145    MDRAW    =$6150
  MDRAW1   =$6155    LOADBUL  =$619A    BDRAW    =$61BD    NOHIT    =$61D9
  COLLISION=$61E1    COL1     =$61E3    BXDRAW   =$61F4    MSHAPE1  =$620C
  MSHAPE2  =$6233    MSHAPE3  =$625A    MSHAPE4  =$6281    MSHAPE5  =$62A8
  MSHAPE6  =$62CF    MSHAPE7  =$62F6    BSHAPE1  =$631D    BSHAPE2  =$631E
  BSHAPE3  =$631F    BSHAPE4  =$6320    BSHAPE5  =$6321    BSHAPE6  =$6322
  BSHAPE7  =$6323    BYTETBL  =$6324    OFFSET   =$6427    HI       =$652A
  LO       =$65EA    GRAPHICS=$C050     MIXOFF   =$C052    PAGE1    =$C054
  HIRES    =$C057    BUTTON   =$C061    PREAD    =$FB1E    WAIT     =$FCA8
```

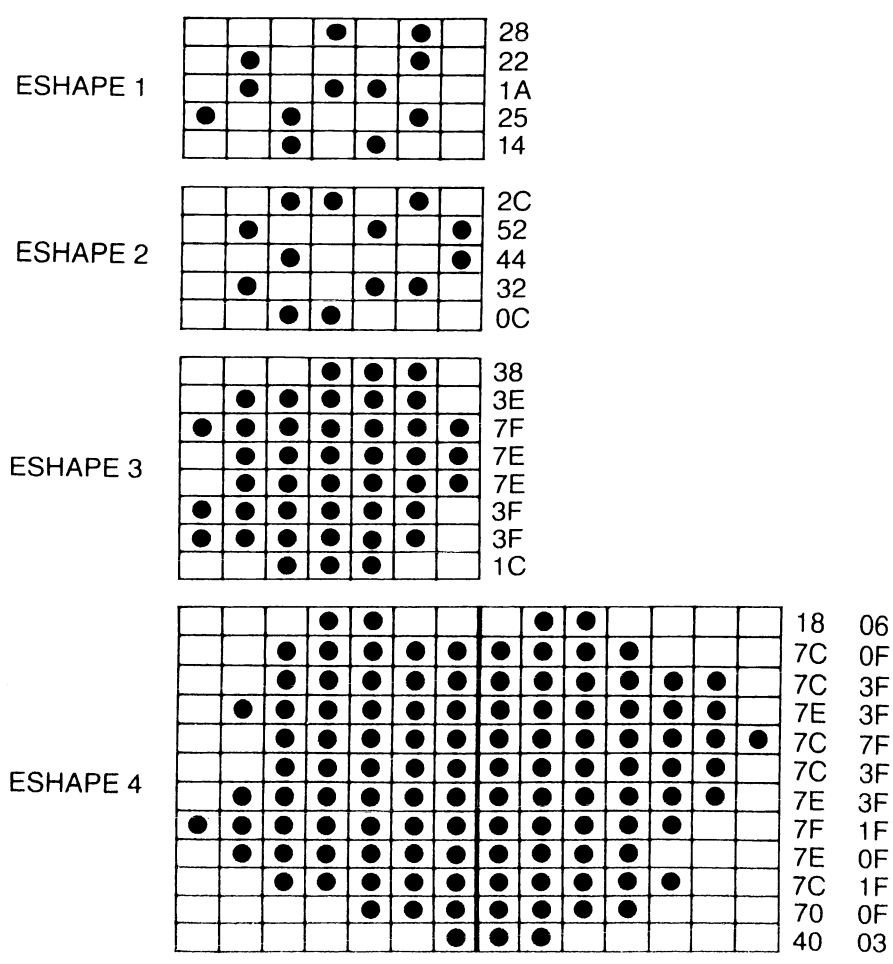

EXPLOSIONS

Collisions don't always result in explosions but they often do (and they will in our final game program), so let's see how we can modify Program 7-1 to display an explosion when a bullet hits the top line (see Program 7-2).

There are two problems associated with explosion routines. One, how do we draw the explosion and two, where do we draw it? Let's tackle the second problem first.

Obviously we want to draw the explosion at the point of impact. How do we determine where this is? Easy. The horizontal position of the explosion is obtained from HORIZB, the horizontal position of the bullet at the time of impact, i.e., when the AND test returns non-zero. The vertical position can be determined from BLINE, the screen line position used for the collision test. Actually, in Program 7-2 and in the final game program, the shape that's hit is always at the same line position. In this case, the vertical position of the impact is known beforehand and we simply can specify this line position in our explosion draw routines. Keep in mind, however, that this is not always the case and so in other situations, BLINE or its equivalent must be used.

For example, suppose we modify the game program so that planes appear at several different line positions—to know where to draw the explosion we would use HORIZB and BLINE. In programs involving collisions with multiple shapes, it's also important to know which shape is hit, because (as we'll see in the game program, although not in the programs in this chapter) the first thing we do after detecting a collision is to erase the target shape. Consider a program where a plane is dropping bombs and we want to detect collisions with both. We know the line position of the plane and so if BLINE tells us we're at that line, we know we've hit the plane. If BLINE tells us the collision is below the plane line, we know we've hit a bomb. Now consider a more complicated example. Suppose we have planes appearing at different lines, each dropping bombs. It's conceivable that a bullet may hit a bomb just at the line position of one of the planes. In this case if we rely just on BLINE, we won't know which shape we've hit. To solve this problem we would use both the bomb and bullet shapes as collision testers. If the bomb and bullet hit something, we know we've hit a bomb. If only the bullet hits something, we know we've hit a plane. Let's take this one step



further. Suppose the bomb hits something but the bullet doesn't. This means the bomb has hit either the bottom line or the man and we can distinguish between these two alternatives by determining at what line the collision took place.

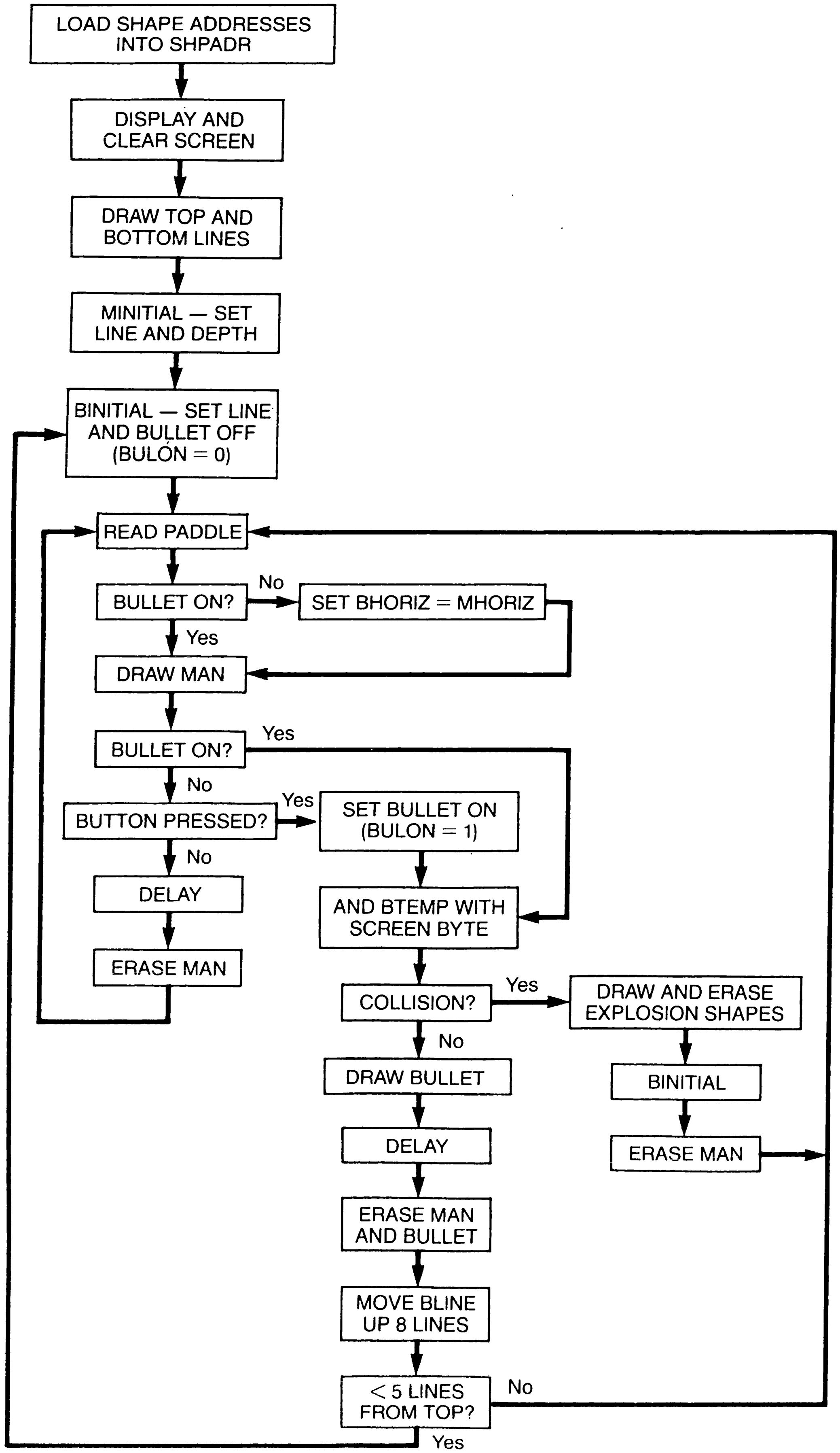
Now let's get to the explosion draw routines. There are many ways to display explosion shapes, from starbursts to splaying of fragments to fireballs, etc. For Program 7-2 and for the game program, we're going to use the fireball type of display. To simulate an explosion, we draw (and erase), at the point of impact, the four shapes (shown on opposite page) in succession—the first two shapes are just random dots, the third is a filled-in "fireball," and the fourth is a larger "fireball."

In Program 7-2, when a collision is detected, the program jumps to the COLLISION subroutine, which then accesses the EXPLOSION subroutine (line 237). Here each of the four shapes in turn is initialized, drawn, delayed, and erased. The program then returns to the COLLISION routine to initialize the bullet, erase the man, and go back for another paddle read.

Each explosion shape has its own initialization routine, labelled INITE1, INITE2, etc., which sets the starting line (ELINE and ELINEA), depth of shape (EDEPTH), and XCOUNT. XCOUNT is specified for each shape because the ESHAPE shape table is accessed in a way that doesn't involve an ESHPADR table, both because it's convenient and because it serves to illustrate that we should not be a slave to any particular type of routine if other routines are equally feasible. In the explosion draw routines, the shape byte is retrieved by EOR ESHAPE,X where X is specified by XCOUNT. Thus, to draw the first shape, we set XCOUNT to zero (lines 323 and 324). Because the first shape contains five bytes, the second shape begins at the sixth position of ESHAPE. Therefore, we set XCOUNT to #$05 for the second shape (ESHAPE + 5 = sixth byte of table). Similarly, we set XCOUNT to #$0A for the third shape and to #$12 for the fourth shape. This type of routine works well if we're dealing with small numbers of shapes and if the shape table is not overly large (we discussed the problem of using this protocol with large shape tables in Chapter 5).

The value for ELINE can be determined from BLINE, the line position of the bullet when the AND test indicates a collision, but as we know where the target is (at screen line #$0C), this becomes unnecessary in this case. However, the particular values we choose for ELINE depend to some extent on how the shape table is constructed and how we want the shapes to be displayed. This is done more or less by trial and error—we simply try different lines to see what looks right. Explosion shapes 1 and 2 are started at line #$09 and because a hot fireball always moves up, shape 3 is started at line #$05 and shape 4 at line #$01, i.e., at higher screen positions.

The explosion draw routines are the usual DRAW-ERASE type except that we have to use two routines, one for the first three shapes (DRAWE1) and another for the fourth shape (DRAWE2), as the fourth shape is 2 bytes wide but the others only 1. For the erase cycle, we first delay and then reinitialize to reset the appropriate parameters—we then go to the draw routine again to erase. The delay times for each shape are also set by trial and error—the ones I've chosen seem to work best. Finally, as mentioned above, the horizontal position of the explosion is determined by HORIZB (see lines 279 and 298).


LOAD SHAPE ADDRESSES INTO SHPADR
DISPLAY AND CLEAR SCREEN
DRAW TOP AND BOTTOM LINES
MINITIAL — SET LINE AND DEPTH
BINITIAL — SET LINE AND BULLET OFF (BULON = 0)
READ PADDLE
BULLET ON?
No
SET BHORIZ = MHORIZ
Yes
DRAW MAN
BULLET ON?
Yes
No
BUTTON PRESSED?
Yes
SET BULLET ON (BULON = 1)
No
DELAY
AND BTEMP WITH SCREEN BYTE
ERASE MAN
COLLISION?
Yes
DRAW AND ERASE EXPLOSION SHAPES
No
DRAW BULLET
BINITIAL
DELAY
ERASE MAN
ERASE MAN AND BULLET
MOVE BLINE UP 8 LINES
< 5 LINES FROM TOP?
No
Yes

```
]PROGRAM 7-2
:ASM
                   *COLLISION AND EXPLOSION*
                            ORG   $6000
6000: 4C 55 60              JMP   PGM
                   MLINE    DS    1
                   MLINEA   DS    1
                   BLINE    DS    1
                   DEPTH    DS    1
                   MHORIZ   DS    1
                   BHORIZ   DS    1
                   HORIZB   DS    1
                   HORIZM   DS    1
                   BULON    DS    1
                   XCOUNT   DS    1
                   DELAY    DS    1
                   BTEMP    DS    1
                   MTEMP    DS    39
                   ELINE    DS    1
                   ELINEA   DS    1
                   EDEPTH   DS    1
                   GRAPHICS =     $C050
                   MIXOFF   =     $C052
                   HIRES    =     $C057
                   PAGE1    =     $C054
                   HIGH     =     $1B
                   LOW      =     $1A
                   WAIT     =     $FCA8
                   PREAD    =     $FB1E
                   BUTTON   =     $C061        ;BUTTON 0
                   *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                   *CONTINUE FOR ALL 7 SHAPES
6039: 09           MSHPADR  DFB   #<MSHAPE1
603A: 63                    DFB   #>MSHAPE1
603B: 30                    DFB   #<MSHAPE2
603C: 63                    DFB   #>MSHAPE2
603D: 57                    DFB   #<MSHAPE3
603E: 63                    DFB   #>MSHAPE3
603F: 7E                    DFB   #<MSHAPE4
6040: 63                    DFB   #>MSHAPE4
6041: A5                    DFB   #<MSHAPE5
6042: 63                    DFB   #>MSHAPE5
6043: CC                    DFB   #<MSHAPE6
6044: 63                    DFB   #>MSHAPE6
6045: F3                    DFB   #<MSHAPE7
6046: 63                    DFB   #>MSHAPE7
6047: 1A           BSHPADR  DFB   #<BSHAPE1
6048: 64                    DFB   #>BSHAPE1
6049: 1B                    DFB   #<BSHAPE2
604A: 64                    DFB   #>BSHAPE2
604B: 1C                    DFB   #<BSHAPE3
604C: 64                    DFB   #>BSHAPE3
604D: 1D                    DFB   #<BSHAPE4
604E: 64                    DFB   #>BSHAPE4
604F: 1E                    DFB   #<BSHAPE5
6050: 64                    DFB   #>BSHAPE5
6051: 1F                    DFB   #<BSHAPE6
6052: 64                    DFB   #>BSHAPE6
6053: 20                    DFB   #<BSHAPE7
6054: 64                    DFB   #>BSHAPE7
6055: AD 50 C0     PGM      LDA   GRAPHICS     ;HIRES,P.1
```

```
6058: AD 52 C0            LDA  MIXOFF
605B: AD 57 C0            LDA  HIRES
605E: AD 54 C0            LDA  PAGE1
6061: A9 00               LDA  #$00         ;CLEAR SCREEN 1
6063: 85 1A               STA  LOW
6065: A9 20               LDA  #$20
6067: 85 1B               STA  HIGH
6069: A0 00      CLR1     LDY  #$00
606B: A9 00               LDA  #$00
606D: 91 1A      CLR      STA  (LOW),Y
606F: C8                  INY
6070: D0 FB               BNE  CLR
6072: E6 1B               INC  HIGH
6074: A5 1B               LDA  HIGH
6076: C9 40               CMP  #$40
6078: 90 EF               BLT  CLR1
607A: A9 50               LDA  #$50         ;LOAD DELAY
607C: 8D 0D 60            STA  DELAY
607F: A2 B7               LDX  #$B7         ;DRAW BOTTOM LINE
6081: A0 00               LDY  #$00
6083: BD 51 66            LDA  HI,X
6086: 85 1B               STA  HIGH
6088: BD 11 67            LDA  LO,X
608B: 85 1A               STA  LOW
608D: A9 7F               LDA  #$7F
608F: 91 1A      LN       STA  (LOW),Y
6091: C8                  INY
6092: C0 27               CPY  #$27
6094: 90 F9               BLT  LN
6096: A2 0C               LDX  #$0C         ;DRAW TOP LINE
6098: A0 00               LDY  #$00
609A: BD 51 66            LDA  HI,X
609D: 85 1B               STA  HIGH
609F: BD 11 67            LDA  LO,X
60A2: 85 1A               STA  LOW
60A4: A9 7F               LDA  #$7F
60A6: 91 1A      LN1      STA  (LOW),Y
60A8: C8                  INY
60A9: C0 14               CPY  #$14
60AB: 90 F9               BLT  LN1
                 ********** MAIN PROGRAM **********
60AD: 20 FE 60            JSR  MINITIAL     ;SET LINE & DEPTH OF MAN
60B0: 20 0D 61            JSR  BINITIAL     ;SET LINE FOR BULLET
60B3: 20 18 61   PADDLE   JSR  PDLE         ;READ PADDLE
60B6: 20 53 61            JSR  MDRAW        ;DRAW MAN
60B9: AD 0B 60            LDA  BULON
60BC: C9 01               CMP  #$01         ;IS BULLET ON?
60BE: F0 16               BEQ  BULLET       ;IF YES, CONTINUE BULLET DRAW
60C0: AD 61 C0            LDA  BUTTON       ;IF NO, IS BUTTON PRESSED?
60C3: 30 0C               BMI  BULLET1      ;IF YES, DRAW BULLET
60C5: AD 0D 60            LDA  DELAY        ;IF NO,
60C8: 20 A8 FC            JSR  WAIT             DELAY AND
60CB: 20 53 61            JSR  MDRAW            ERASE MAN AND
60CE: 4C B3 60            JMP  PADDLE           READ PADDLE AGAIN
60D1: A9 01      BULLET1  LDA  #$01         ;SET BULLET ON
60D3: 8D 0B 60            STA  BULON
60D6: 20 9D 61   BULLET   JSR  LOADBUL      ;LOAD BULLET SHAPE INTO BTEMP
60D9: 20 C0 61            JSR  BDRAW        ;DRAW BULLET & TEST FOR COLLISION
60DC: AD 0D 60            LDA  DELAY
60DF: 20 A8 FC            JSR  WAIT         ;DELAY
60E2: 20 F0 61            JSR  BXDRAW       ;ERASE BULLET
```

```
60E5: 20 53 61          JSR  MDRAW      ;ERASE MAN
60E8: AD 05 60          LDA  BLINE
60EB: 38                SEC
60EC: E9 08             SBC  #$08       ;MOVE BLINE UP 8 LINES
60EE: 8D 05 60          STA  BLINE
60F1: C9 05             CMP  #$05       ;LESS THAN 5 LINES FROM TOP?
60F3: 90 03             BLT  TOP        ;IF YES TAKE BRANCH
60F5: 4C B3 60          JMP  PADDLE     ;IF NO, READ PADDLE AGAIN
60F8: 20 0D 61 TOP      JSR  BINITIAL   ;INITIALIZE BULLET LINE
60FB: 4C B3 60          JMP  PADDLE     ;READ PADDLE
               ********** SUBROUTINES **********
60FE: A9 AA    MINITIAL LDA  #$AA
6100: 8D 03 60          STA  MLINE
6103: 8D 04 60          STA  MLINEA
6106: 18                CLC
6107: 69 0D             ADC  #$0D
6109: 8D 06 60          STA  DEPTH
610C: 60                RTS
               ***********************
610D: A9 00    BINITIAL LDA  #$00       ;BULON = 0 IF
610F: 8D 0B 60          STA  BULON          BULLET NOT ON SCREEN
6112: A9 A4             LDA  #$A4
6114: 8D 05 60          STA  BLINE
6117: 60                RTS
               **********************
6118: A2 00    PDLE     LDX  #$00
611A: 20 1E FB          JSR  PREAD      ;READ PADDLE 0
611D: 98                TYA
611E: 8D 07 60          STA  MHORIZ     ;0-255 IN MHORIZ
6121: AD 0B 60          LDA  BULON
6124: C9 01             CMP  #$01       ;IS BULLET ON?
6126: F0 06             BEQ  PDLE1      ;IF YES, TAKE BRANCH
6128: AD 07 60          LDA  MHORIZ     ;IF NO, SET BHORIZ EQUAL
612B: 8D 08 60          STA  BHORIZ         TO MHORIZ
612E: AC 07 60 PDLE1    LDY  MHORIZ
6131: B9 4B 64          LDA  BYTETBL,Y  ;CONVERT 0-255 TO 0-36 (BYTE)
6134: 8D 0A 60          STA  HORIZM     ;MAN BYTE POSITION
6137: B9 4E 65          LDA  OFFSET,Y   ;GET SHAPE NUMBER
613A: 0A                ASL             ;LOAD SHAPE INTO MTEMP
613B: AA                TAX
613C: BD 39 60          LDA  MSHPADR,X
613F: 85 1A             STA  LOW
6141: BD 3A 60          LDA  MSHPADR+1,X
6144: 85 1B             STA  HIGH
6146: A0 00             LDY  #$00
6148: B1 1A    LOAD     LDA  (LOW),Y
614A: 99 0F 60          STA  MTEMP,Y
614D: C8                INY
614E: C0 27             CPY  #$27
6150: 90 F6             BLT  LOAD
6152: 60                RTS
               ***********************
6153: A9 00    MDRAW    LDA  #$00
6155: 8D 0C 60          STA  XCOUNT
6158: AE 03 60 MDRAW1   LDX  MLINE
615B: AC 0A 60          LDY  HORIZM
615E: BD 51 66          LDA  HI,X
6161: 85 1B             STA  HIGH
6163: BD 11 67          LDA  LO,X
6166: 85 1A             STA  LOW
6168: AE 0C 60          LDX  XCOUNT
```

```
616B: B1 1A              LDA  (LOW),Y
616D: 5D 0F 60           EOR  MTEMP,X
6170: 91 1A              STA  (LOW),Y
6172: C8                 INY
6173: B1 1A              LDA  (LOW),Y
6175: 5D 10 60           EOR  MTEMP+1,X
6178: 91 1A              STA  (LOW),Y
617A: C8                 INY
617B: B1 1A              LDA  (LOW),Y
617D: 5D 11 60           EOR  MTEMP+2,X
6180: 91 1A              STA  (LOW),Y
6182: EE 0C 60           INC  XCOUNT
6185: EE 0C 60           INC  XCOUNT
6188: EE 0C 60           INC  XCOUNT
618B: EE 03 60           INC  MLINE
618E: AD 03 60           LDA  MLINE
6191: CD 06 60           CMP  DEPTH
6194: 90 C2              BLT  MDRAW1
6196: AD 04 60           LDA  MLINEA      ;RESET LINE
6199: 8D 03 60           STA  MLINE
619C: 60                 RTS
                ***********************
619D: AC 08 60  LOADBUL  LDY  BHORIZ      ;CONVERTS 0-255 TO
61A0: B9 4B 64           LDA  BYTETBL,Y       SCREEN BYTE (0-36)
61A3: 18                 CLC              ;ADD 2 TO ALIGN BULLET
61A4: 69 02              ADC  #$02            WITH GUN
61A6: 8D 09 60           STA  HORIZB      ;BULLET BYTE POSITION
61A9: B9 4E 65           LDA  OFFSET,Y    ;GET BULLET SHAPE NUMBER
61AC: 0A                 ASL              ;LOAD BULLET SHAPE INTO BTEMP
61AD: AA                 TAX
61AE: BD 47 60           LDA  BSHPADR,X
61B1: 85 1A              STA  LOW
61B3: BD 48 60           LDA  BSHPADR+1,X
61B6: 85 1B              STA  HIGH
61B8: A0 00              LDY  #$00
61BA: B1 1A              LDA  (LOW),Y
61BC: 8D 0E 60           STA  BTEMP
61BF: 60                 RTS
                ***********************
61C0: AE 05 60  BDRAW    LDX  BLINE
61C3: AC 09 60           LDY  HORIZB
61C6: BD 51 66           LDA  HI,X
61C9: 85 1B              STA  HIGH
61CB: BD 11 67           LDA  LO,X
61CE: 85 1A              STA  LOW
61D0: B1 1A              LDA  (LOW),Y
61D2: 2D 0E 60           AND  BTEMP       ;RESULT IS 0 IF NO COLLISION
61D5: C9 00              CMP  #$00
61D7: F0 03              BEQ  NOHIT
61D9: 4C E4 61           JMP  COLLISION
61DC: B1 1A     NOHIT    LDA  (LOW),Y     ;DRAW BULLET
61DE: 4D 0E 60           EOR  BTEMP
61E1: 91 1A              STA  (LOW),Y
61E3: 60                 RTS
                ************************
61E4: 20 08 62  COLLISION JSR EXPLODE
61E7: 20 0D 61           JSR  BINITIAL
61EA: 20 53 61           JSR  MDRAW       ;ERASE MAN
61ED: 4C B3 60           JMP  PADDLE
                ************************
61F0: AE 05 60  BXDRAW   LDX  BLINE       ;BDRAW WITHOUT COLLISION TEST
```

```
61F3: AC 09 60                LDY  HORIZB
61F6: BD 51 66                LDA  HI,X
61F9: 85 1B                   STA  HIGH
61FB: BD 11 67                LDA  LO,X
61FE: 85 1A                   STA  LOW
6200: B1 1A                   LDA  (LOW),Y
6202: 4D 0E 60                EOR  BTEMP
6205: 91 1A                   STA  (LOW),Y
6207: 60                      RTS
                   ***********************
6208: 20 B9 62       EXPLODE  JSR  INITE1
620B: 20 4D 62                JSR  DRAWE1      ;DRAW
620E: A9 60                   LDA  #$60
6210: 20 A8 FC                JSR  WAIT
6213: 20 B9 62                JSR  INITE1
6216: 20 4D 62                JSR  DRAWE1      ;ERASE
6219: 20 CD 62                JSR  INITE2
621C: 20 4D 62                JSR  DRAWE1      ;DRAW
621F: A9 BB                   LDA  #$BB
6221: 20 A8 FC                JSR  WAIT
6224: 20 CD 62                JSR  INITE2
6227: 20 4D 62                JSR  DRAWE1      ;ERASE
622A: 20 E1 62                JSR  INITE3
622D: 20 4D 62                JSR  DRAWE1      ;DRAW
6230: A9 BB                   LDA  #$BB
6232: 20 A8 FC                JSR  WAIT
6235: 20 E1 62                JSR  INITE3
6238: 20 4D 62                JSR  DRAWE1      ;ERASE
623B: 20 F5 62                JSR  INITE4
623E: 20 7C 62                JSR  DRAWE2      ;DRAW
6241: A9 FF                   LDA  #$FF
6243: 20 A8 FC                JSR  WAIT
6246: 20 F5 62                JSR  INITE4
6249: 20 7C 62                JSR  DRAWE2      ;ERASE
624C: 60                      RTS
                   *************************
624D: AC 09 60       DRAWE1   LDY  HORIZB      ;ROUTINE FOR FIRST 3
6250: AE 36 60                LDX  ELINE           EXPLOSION SHAPES
6253: BD 51 66                LDA  HI,X
6256: 85 1B                   STA  HIGH
6258: BD 11 67                LDA  LO,X
625B: 85 1A                   STA  LOW
625D: AE 0C 60                LDX  XCOUNT
6260: B1 1A                   LDA  (LOW),Y
6262: 5D 21 64                EOR  ESHAPE,X
6265: 91 1A                   STA  (LOW),Y
6267: EE 0C 60                INC  XCOUNT
626A: EE 36 60                INC  ELINE
626D: AD 36 60                LDA  ELINE
6270: CD 38 60                CMP  EDEPTH
6273: 90 D8                   BLT  DRAWE1
6275: AD 37 60                LDA  ELINEA
6278: 8D 36 60                STA  ELINE
627B: 60                      RTS
                   ***********************
627C: AC 09 60       DRAWE2   LDY  HORIZB      ;ROUTINE FOR FOURTH
627F: AE 36 60                LDX  ELINE           EXPLOSION SHAPE
6282: BD 51 66                LDA  HI,X
6285: 85 1B                   STA  HIGH
6287: BD 11 67                LDA  LO,X
628A: 85 1A                   STA  LOW
```

```
628C: AE 0C 60             LDX  XCOUNT
628F: B1 1A                LDA  (LOW),Y
6291: 5D 21 64             EOR  ESHAPE,X
6294: 91 1A                STA  (LOW),Y
6296: EE 0C 60             INC  XCOUNT
6299: C8                   INY
629A: AE 0C 60             LDX  XCOUNT
629D: B1 1A                LDA  (LOW),Y
629F: 5D 21 64             EOR  ESHAPE,X
62A2: 91 1A                STA  (LOW),Y
62A4: EE 0C 60             INC  XCOUNT
62A7: EE 36 60             INC  ELINE
62AA: AD 36 60             LDA  ELINE
62AD: CD 38 60             CMP  EDEPTH
62B0: 90 CA                BLT  DRAWE2
62B2: AD 37 60             LDA  ELINEA
62B5: 8D 36 60             STA  ELINE
62B8: 60                   RTS
                   ************************
62B9: A9 00        INITE1  LDA  #$00         ;INITIALIZE FIRST EXPLOSION
62BB: 8D 0C 60             STA  XCOUNT
62BE: A9 09                LDA  #$09
62C0: 8D 37 60             STA  ELINEA
62C3: 8D 36 60             STA  ELINE
62C6: 18                   CLC
62C7: 69 05                ADC  #$05
62C9: 8D 38 60             STA  EDEPTH
62CC: 60                   RTS
62CD: A9 05        INITE2  LDA  #$05         ;INITIALIZE SECOND EXPLOSION
62CF: 8D 0C 60             STA  XCOUNT
62D2: A9 09                LDA  #$09
62D4: 8D 37 60             STA  ELINEA
62D7: 8D 36 60             STA  ELINE
62DA: 18                   CLC
62DB: 69 05                ADC  #$05
62DD: 8D 38 60             STA  EDEPTH
62E0: 60                   RTS
62E1: A9 0A        INITE3  LDA  #$0A         ;INITIALIZE THIRD EXPLOSION
62E3: 8D 0C 60             STA  XCOUNT
62E6: A9 05                LDA  #$05
62E8: 8D 37 60             STA  ELINEA
62EB: 8D 36 60             STA  ELINE
62EE: 18                   CLC
62EF: 69 08                ADC  #$08
62F1: 8D 38 60             STA  EDEPTH
62F4: 60                   RTS
62F5: A9 12        INITE4  LDA  #$12         ;INITIALIZE FOURTH EXPLOSION
62F7: 8D 0C 60             STA  XCOUNT
62FA: A9 01                LDA  #$01
62FC: 8D 37 60             STA  ELINEA
62FF: 8D 36 60             STA  ELINE
6302: 18                   CLC
6303: 69 0C                ADC  #$0C
6305: 8D 38 60             STA  EDEPTH
6308: 60                   RTS
                   ************************
6309: 00 0E 01     MSHAPE1 HEX  000E01000E01000E01   ;MAN SHAPE TABLES
630C: 00 0E 01 00 0E 01
6312: 00 44 01             HEX  004401007F00601F00
6315: 00 7F 00 60 1F 00
631B: 30 1F 00             HEX  301F00181F00001F00
```

```
631E: 18 1F 00 00 1F 00
6324: 00 1F 00  363             HEX  001F00001B00403100
6327: 00 1B 00 40 31 00
632D: 60 60 00  364             HEX  606000
6330: 00 1C 02  365  MSHAPE2    HEX  001C02001C02001C02
6333: 00 1C 02 00 1C 02
6339: 00 08 03  366             HEX  000803007E01003E00
633C: 00 7E 01 00 3E 00
6342: 00 3F 00  367             HEX  003F00403F00003E00
6345: 40 3F 00 00 3E 00
634B: 00 3E 00  368             HEX  003E00003600003600
634E: 00 36 00 00 36 00
6354: 00 63 00  369             HEX  006300
6357: 00 38 04  370  MSHAPE3    HEX  003804003804003804
635A: 00 38 04 00 38 04
6360: 00 10 06  371             HEX  001006007C03007C00
6363: 00 7C 03 00 7C 00
6369: 00 7C 00  372             HEX  007C00007E00007C00
636C: 00 7E 00 00 7C 00
6372: 00 38 00  373             HEX  003800003800006C00
6375: 00 38 00 00 6C 00
637B: 00 46 01  374             HEX  004601
637E: 00 70 08  375  MSHAPE4    HEX  007008007008007008
6381: 00 70 08 00 70 08
6387: 00 20 0C  376             HEX  00200C007807007801
638A: 00 78 07 00 78 01
6390: 00 78 01  377             HEX  007801007801007801
6393: 00 78 01 00 78 01
6399: 00 70 00  378             HEX  007000007000007000
639C: 00 70 00 00 70 00
63A2: 00 70 00  379             HEX  007000
63A5: 00 60 11  380  MSHAPE5    HEX  006011006011006011
63A8: 00 60 11 00 60 11
63AE: 00 40 18  381             HEX  00401800700F007003
63B1: 00 70 0F 00 70 03
63B7: 00 70 03  382             HEX  007003007803007003
63BA: 00 78 03 00 70 03
63C0: 00 60 01  383             HEX  006001006001003003
63C3: 00 60 01 00 30 03
63C9: 00 18 06  384             HEX  001806
63CC: 00 40 23  385  MSHAPE6    HEX  004023004023004023
63CF: 00 40 23 00 40 23
63D5: 00 00 31  386             HEX  00003100601F006007
63D8: 00 60 1F 00 60 07
63DE: 00 70 07  387             HEX  007007007807006007
63E1: 00 78 07 00 60 07
63E7: 00 60 07  388             HEX  006007006006006006
63EA: 00 60 06 00 60 06
63F0: 00 30 0C  389             HEX  00300C
63F3: 00 00 47  390  MSHAPE7    HEX  000047000047000047
63F6: 00 00 47 00 00 47
63FC: 00 00 62  391             HEX  00006200403F00700F
63FF: 00 40 3F 00 70 0F
6405: 00 58 0F  392             HEX  00580F004C0F00400F
6408: 00 4C 0F 00 40 0F
640E: 00 40 0F  393             HEX  00400F00400D006018
6411: 00 40 0D 00 60 18
6417: 00 30 30  394             HEX  003030
641A: 01        395  BSHAPE1    HEX  01                ;BULLET SHAPES
641B: 02        396  BSHAPE2    HEX  02
641C: 04        397  BSHAPE3    HEX  04
```

```
641D: 08          398  BSHAPE4   HEX  08
641E: 10          399  BSHAPE5   HEX  10
641F: 20          400  BSHAPE6   HEX  20
6420: 40          401  BSHAPE7   HEX  40
6421: 28 22 1A    402  ESHAPE    HEX  28221A2514      ;EXPLOSION SHAPES - NO. 1
6424: 25 14
6426: 2C 52 44    403            HEX  2C5244320C        ;NO. 2
6429: 32 0C
642B: 38 3E 7F    404            HEX  383E7F7E7E3F3F1C  ;NO. 3
642E: 7E 7E 3F 3F 1C
6433: 18 06 7C    405            HEX  18067C0F7C3F7E3F  ;NO. 4
6436: 0F 7C 3F 7E 3F
643B: 7C 7F 7C    406            HEX  7C7F7C3F7E3F7F1F
643E: 3F 7E 3F 7F 1F
6443: 7E 0F 7C    407            HEX  7E0F7C1F700F4003
6446: 1F 70 0F 40 03
                       BYTETBL
                       OFFSET
                       HI
                       LO

2001 bytes

Symbol table - numerical order:

  LOW      =$1A     HIGH     =$1B     MLINE    =$6003    MLINEA   =$6004
  BLINE    =$6005   DEPTH    =$6006   MHORIZ   =$6007    BHORIZ   =$6008
  HORIZB   =$6009   HORIZM   =$600A   BULON    =$600B    XCOUNT   =$600C
  DELAY    =$600D   BTEMP    =$600E   MTEMP    =$600F    ELINE    =$6036
  ELINEA   =$6037   EDEPTH   =$6038   MSHPADR  =$6039    BSHPADR  =$6047
  PGM      =$6055   CLR1     =$6069   CLR      =$606D    LN       =$608F
  LN1      =$60A6   PADDLE   =$60B3   BULLET1  =$60D1    BULLET   =$60D6
  TOP      =$60F8   MINITIAL=$60FE    BINITIAL=$610D     PDLE     =$6118
  PDLE1    =$612E   LOAD     =$6148   MDRAW    =$6153    MDRAW1   =$6158
  LOADBUL  =$619D   BDRAW    =$61C0   NOHIT    =$61DC    COLLISION=$61E4
  BXDRAW   =$61F0   EXPLODE  =$6208   DRAWE1   =$624D    DRAWE2   =$627C
  INITE1   =$62B9   INITE2   =$62CD   INITE3   =$62E1    INITE4   =$62F5
  MSHAPE1  =$6309   MSHAPE2  =$6330   MSHAPE3  =$6357    MSHAPE4  =$637E
  MSHAPE5  =$63A5   MSHAPE6  =$63CC   MSHAPE7  =$63F3    BSHAPE1  =$641A
  BSHAPE2  =$641B   BSHAPE3  =$641C   BSHAPE4  =$641D    BSHAPE5  =$641E
  BSHAPE6  =$641F   BSHAPE7  =$6420   ESHAPE   =$6421    BYTETBL  =$644B
  OFFSET   =$654E   HI       =$6651   LO       =$6711    GRAPHICS=$C050
  MIXOFF   =$C052   PAGE1    =$C054   HIRES    =$C057    BUTTON   =$C061
  PREAD    =$FB1E   WAIT     =$FCA8
```


One last point. Run Program 7-2 and pay particular attention to the paddle control of the man movement while the explosion shapes are being drawn. You will see, if you look carefully enough, that the man becomes unresponsive to the paddle control until the explosion shapes are finished. An examination of the Program 7-2 flowchart will tell you why this is happening. The entire explosion routine is run before the program branches back for another paddle read. Because the explosion routine uses up some amount of time caused by all the delays between shapes, the program is interrupted momentarily. In many cases, however, including this one and the game program, such an interruption is acceptable because it is of fairly short duration and the man is not moving at some constant speed and thus doesn't appear to "freeze" during the delay. In fact, the user's attention would probably be riveted on the explosion rather than on trying to move the man and so would hardly notice the unresponsiveness of the paddle (it's hardly noticeable even when you're paying attention).

Suppose, however, we have two plane shapes moving across the screen at the same time. When one is hit, the other will freeze in position until the explosion shapes have all been drawn and erased. This would be noticeable and should be avoided. We can solve this problem in the following way. In the EXPLODE subroutine, instead of inserting a delay between each draw and erase, we branch to a routine that will erase the man and read the paddle for another man draw and then return to the explosion. In other words, a program loop will replace the delays. The protocol would then be draw-loop-erase-draw-loop-erase, etc. The loop cannot use the same paddle read routine that's already in the program because we don't want to go through another bullet draw and collision test, and so we would have another routine that would consist of just erase man—paddle read—draw man. The time for the loop will almost assuredly not be as long as the delays already in the explosion routine and so we would probably have to introduce some time delay between the draw and erase cycles. The exact delays to insert would again be a matter of trial and error—we would just try different values until that second plane moved properly.

This discussion emphasizes an important point about game design (or for that matter any program that is doing more than one thing at a time on the screen) and that is the time delays that are necessary to reduce shape flicker (and sometimes to slow the program down) are exactly the points where one can introduce program loops when one is expanding the program to do more things. We will see, for example, when we get to the game program, that an explosion sound routine is inserted in place of the delay between drawing and erasing the first explosion shape. The sound routine itself provides the necessary delay, and we end up with a program where the explosion looks the same as before, but now with an extra feature.

8

Scoring, Stopping, and Restarting

A salesman with motives deplorable
Showed an Apple game to a customer adorable.
He said, "This game we'll now play
Scores every which way,
But scoring with you is preferable."

Scoring in game programs is almost a law of nature. I can't think of one game program I've tried over the years that didn't have some type of score routine. Of course now that we're experts in hi-res graphics, we can devise any type of scoring display we desire. We can choose our own number shapes, put them anywhere on the screen we want, enclose them in a scoring box with a title, even count in Roman numerals! But let's not get too ambitious. The first program in this chapter discusses a simple scoring routine that's used in the game program. The second program presents a routine of more general utility. We're also going to discuss how to stop a program at a predetermined score and how to start it again from the keyboard.

COUNTING BY ONES

In the following program (Program 8-1), we're going to modify Program 7-2 so that a score display, initially set to 000, increments by 1 each time the top line is hit by a bullet. When the score reaches 100, the program stops and can be restarted by pressing any key. The scoring routine in this program will be incorporated into the game program.

The numbers we'll be using to display the score are simply hi-res shapes depicting digits 0 to 9 (big surprise, eh?). Each number shape is 1 byte wide by 8 lines deep and the 10 shapes are stored in a table labeled NSHAPE. These shapes are accessed and printed in a way we haven't seen before, just for variety's sake and to show off our assembly language dexterity. In the PRINT subroutine, the beginning of each number shape is accessed by LDA NSHAPE,X. Because each number contains 8 bytes, when X = 0, the beginning of the first shape (digit 0) is accessed, when X = 8, the second shape (digit 1) is accessed, when X = 16, the third shape (digit 2 is accessed), etc. Once X is specified, PRINT then accesses each byte of the designated shape in turn by LDA NSHAPE+1,X, LDA NSHAPE+2,X ...... LDA NSHAPE+7,X. As each byte of the number shape is retrieved, it is printed on a separate line, starting from line 184 to line 191, i.e., just below the bottom line the man is walking on.

The lines where the shapes are to be drawn are specified as direct addresses from the hi-res screen memory map instead of from the line address tables—this saves execution time and program space and is easy to do when dealing with small routines such as PRINT. (In fact, in some programs, if they're large and complicated enough, the use of direct addresses may be called for just to get the program to run fast enough, as a table look-up is a time-consuming process.) The draw instruction has the form STA $23D0,Y (this is for line 184)—$23D0 specifies the line, whereas Y specifies the horizontal position where the byte is to be drawn.

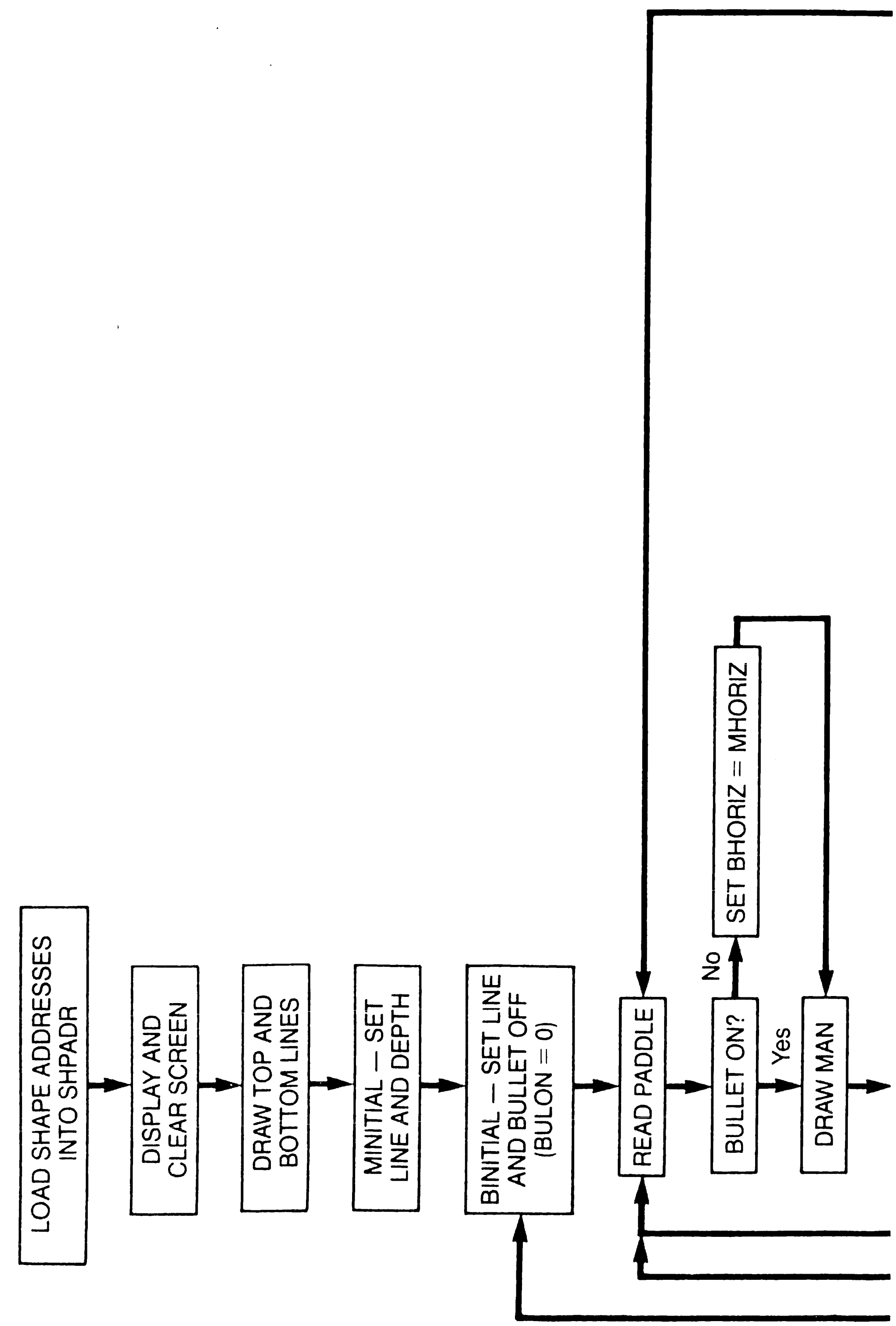
To summarize—X specifies the number shape from 0 to 9, the line positions are specified directly in PRINT, and Y specifies the horizontal position.

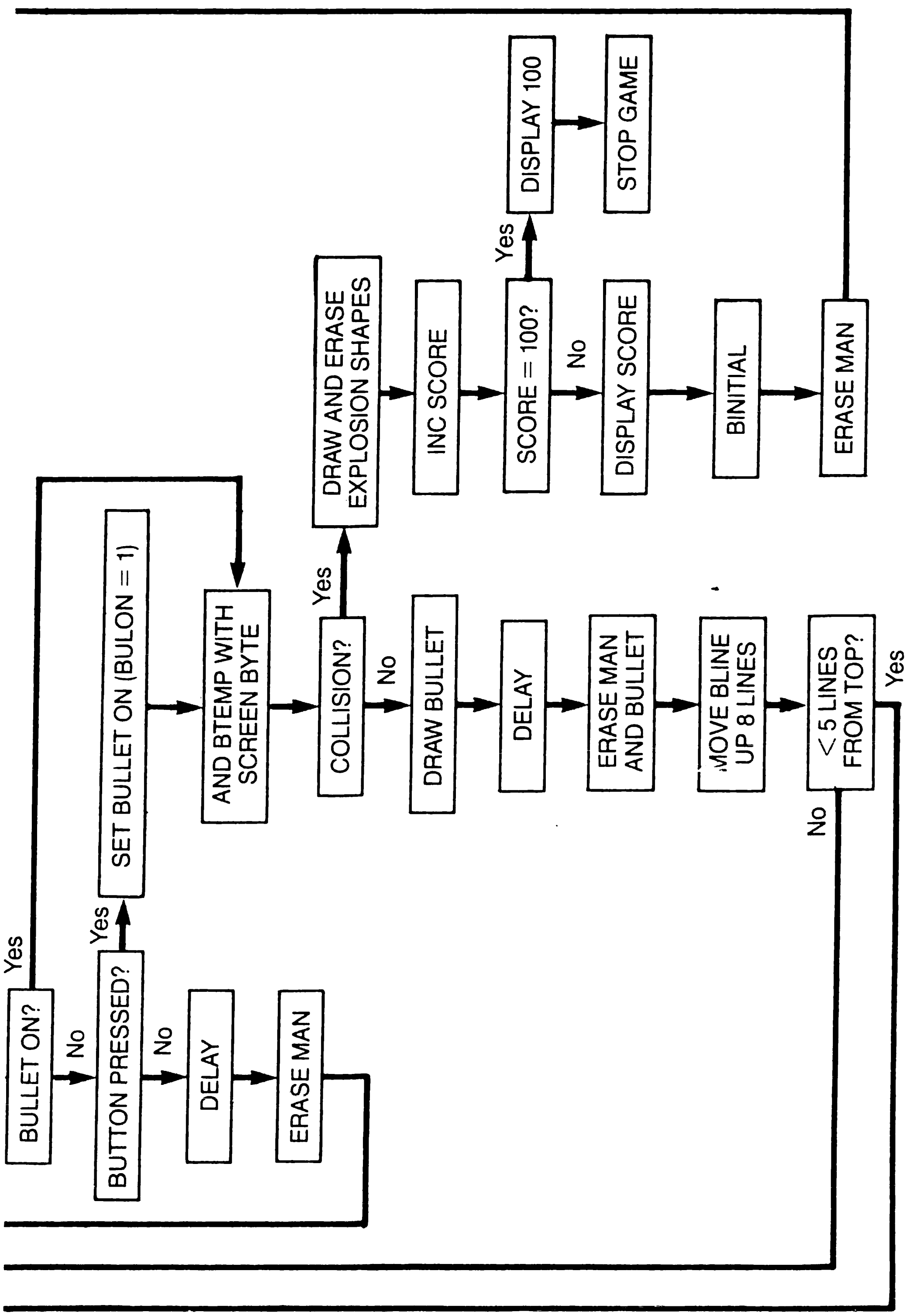
To see how all this works, let's look at the SINITIAL subroutine that prints 000 just below the bottom line at the center of the screen. This is done at the beginning of the program (line 105) to zero the counter. First, the Accumulator is loaded with #$00 (line 149). Memory locations labelled SUM and COUNTER are also zeroed (we'll get to these later). The next instruction, TAX (line 152), transfers the contents of the Accumulator, #$00, to the X register. Because X = 0, when we go to the PRINT subroutine the first number shape (digit 0) will be printed. Line 153 loads Y with #$11, the horizontal position of the first or left-most digit in the counter display. JSR PRINT then prints 0 at that position. Y is then incremented and another 0 is printed at position #$12. Finally, a third 0 is printed at #$13—the loop stops when Y = #$14. The relationship of Y then to the counter digits is as follows:

#$11	#$12	#$13
0	0	0

Now that we've zeroed the counter, let's see how we count collision events. Every time a collision is detected, we do a INC SUM and JSR SCORE (lines 252 and 253) in the COLLISION subroutine. In the SCORE subroutine, we load the Accumulator with SUM and compare the value to 10 (#$0A). If it's not equal to 10, we skip the branch in line 378 and multiply the number by 8. (Remember ASL multiplies by 2 and so three ASLs gives 2 X 2 X 2 = 8; if you don't understand this, return this book for an immediate refund.) We multiply by 8 to access the proper number shape. Thus, when SUM = 1, X = 8 and digit 1 is accessed; when SUM = 2, X = 16 and digit 2 is accessed, and so on. We then transfer the number to X, load Y with #$13, and print with a JSR PRINT. The number is printed in the rightmost counter position so the counter will now display 001, 002, etc.

Suppose now the number in SUM has been incremented to 10. The branch at line 378 is taken and we go to C10 to increment COUNTER and load the Accumulator with COUNTER (lines 386 and 387). If the number in COUNTER is not equal to 10, the branch in line 389 is not taken and the number in COUNTER is multiplied by 8, transferred to X, and printed at position #$12 (Y = #$12, line 394). The middle counter digit will then be 1. We then zero SUM and jump back to SCORE to print a zero in the rightmost position. The counter now


LOAD SHAPE ADDRESSES INTO SHPADR
DISPLAY AND CLEAR SCREEN
DRAW TOP AND BOTTOM LINES
MINITIAL — SET LINE AND DEPTH
BINITIAL — SET LINE AND BULLET OFF (BULON = 0)
READ PADDLE
BULLET ON?
No
SET BHORIZ = MHORIZ
Yes
DRAW MAN


BULLET ON?
Yes
No
BUTTON PRESSED?
Yes
No
DELAY
ERASE MAN
SET BULLET ON (BULON = 1)
AND BTEMP WITH SCREEN BYTE
COLLISION?
Yes
No
DRAW BULLET
DELAY
ERASE MAN AND BULLET
MOVE BLINE UP 8 LINES
< 5 LINES FROM TOP?
Yes
No
DRAW AND ERASE EXPLOSION SHAPES
INC SCORE
SCORE = 100?
Yes
No
DISPLAY 100
STOP GAME
DISPLAY SCORE
BINITIAL
ERASE MAN

displays 010, which is what we want because SUM = 10. Now when we increment again, SUM will contain 1 and a 1 will be printed at Y = #$13. Because SUM is again less than 10, the branch to C10 is not taken and COUNTER retains its value of 1. Thus, the counter will display 011. The counting then continues. Each time SUM gets to 10, COUNTER is incremented by 1 and SUM is zeroed; COUNTER is printed in Y = #$12 and SUM in Y = #$13.

With this routine we can count to essentially any number simply by specifying other counters, such as COUNTER1 for the 100s column, COUNTER2 for the 1000s column, etc., and modifying the routine to access these counters at the appropriate times; e.g., when COUNTER reaches 10, COUNTER1 is incremented by 1 and printed in #$11, and so on. Also, we're not limited by the fact that a memory location can hold a maximum value of #$FF, as SUM and COUNTER never contain values greater than 10.

STOPPING AT A PREDETERMINED SCORE AND RESTARTING WITH A KEYPRESS

SUM is the counter for the 1's column and COUNTER for the 10's column. Thus, when COUNTER gets to 10, it means the score has reached 100. The branch at line 389 is then taken and the program jumps to STOP1. Here 100 is printed in the counter display and the program then goes to STOP. At this routine, the instruction BIT $C000 accesses a soft switch that says watch for any keypress. BIT—Compare Accumulator BITS with contents of memory—is an instruction whose main functions I simply do not want to get into and you wouldn't want to either if you saw what it does. Maybe in some future book on advanced techniques (Volume DCMXIII?) I'll discuss it, but for now I use it only to illustrate one of its more arcane but useful features—it can replace LDA or STA to access a soft switch, and it does so without damaging the contents of the Accumulator.

To get back to line 407, BIT $C000 says watch for any keypress and the next line (BPL STOP) says if no key is pressed, go back and watch again. This loop continues until any key is pressed, at which point the program continues to BIT $C010, which accesses a soft switch to clear the keyboard strobe (the keyboard holds the last key pressed until either another key is pressed or until the strobe is cleared), and then finally to JMP PGM which starts the program over. Note that we don't have to go back to the program starting line at $6000 for a restart because everything from $6000 to PGM has already been done and is in memory. (See flowchart on pages 142 and 143.)

```
]PROGRAM 8-1
:ASM
                1    *COLLISION AND EXPLOSION WITH SCORING*
                2             ORG  $6000
6000: 4C 57 60  3             JMP  PGM
                4    MLINE    DS   1
                5    MLINEA   DS   1
                6    BLINE    DS   1
                7    DEPTH    DS   1
                8    MHORIZ   DS   1
                9    BHORIZ   DS   1
                10   HORIZB   DS   1
                11   HORIZM   DS   1
                12   BULON    DS   1
                13   XCOUNT   DS   1
```

```
                     DELAY    DS   1
                     BTEMP    DS   1
                     MTEMP    DS   39
                     ELINE    DS   1
                     ELINEA   DS   1
                     EDEPTH   DS   1
                     SUM      DS   1
                     COUNTER  DS   1
                     GRAPHICS =    $C050
                     MIXOFF   =    $C052
                     HIRES    =    $C057
                     PAGE1    =    $C054
                     HIGH     =    $1B
                     LOW      =    $1A
                     WAIT     =    $FCA8
                     PREAD    =    $FB1E
                     BUTTON   =    $C061      ;BUTTON 0
                     *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                     *CONTINUE FOR ALL 7 SHAPES
603B: F3             MSHPADR  DFB  #<MSHAPE1
603C: 63                      DFB  #>MSHAPE1
603D: 1A                      DFB  #<MSHAPE2
603E: 64                      DFB  #>MSHAPE2
603F: 41                      DFB  #<MSHAPE3
6040: 64                      DFB  #>MSHAPE3
6041: 68                      DFB  #<MSHAPE4
6042: 64                      DFB  #>MSHAPE4
6043: 8F                      DFB  #<MSHAPE5
6044: 64                      DFB  #>MSHAPE5
6045: B6                      DFB  #<MSHAPE6
6046: 64                      DFB  #>MSHAPE6
6047: DD                      DFB  #<MSHAPE7
6048: 64                      DFB  #>MSHAPE7
6049: 04             BSHPADR  DFB  #<BSHAPE1
604A: 65                      DFB  #>BSHAPE1
604B: 05                      DFB  #<BSHAPE2
604C: 65                      DFB  #>BSHAPE2
604D: 06                      DFB  #<BSHAPE3
604E: 65                      DFB  #>BSHAPE3
604F: 07                      DFB  #<BSHAPE4
6050: 65                      DFB  #>BSHAPE4
6051: 08                      DFB  #<BSHAPE5
6052: 65                      DFB  #>BSHAPE5
6053: 09                      DFB  #<BSHAPE6
6054: 65                      DFB  #>BSHAPE6
6055: 0A                      DFB  #<BSHAPE7
6056: 65                      DFB  #>BSHAPE7
6057: AD 50 C0       PGM      LDA  GRAPHICS   ;HIRES,P.1
605A: AD 52 C0                LDA  MIXOFF
605D: AD 57 C0                LDA  HIRES
6060: AD 54 C0                LDA  PAGE1
6063: A9 00                   LDA  #$00       ;CLEAR SCREEN 1
6065: 85 1A                   STA  LOW
6067: A9 20                   LDA  #$20
6069: 85 1B                   STA  HIGH
606B: A0 00          CLR1     LDY  #$00
606D: A9 00                   LDA  #$00
606F: 91 1A          CLR      STA  (LOW),Y
6071: C8                      INY
6072: D0 FB                   BNE  CLR
6074: E6 1B                   INC  HIGH
```

```
6076: A5 1B              LDA  HIGH
6078: C9 40              CMP  #$40
607A: 90 EF              BLT  CLR1
607C: A9 50              LDA  #$50       ;LOAD DELAY
607E: 8D 0D 60           STA  DELAY
6081: A2 B7              LDX  #$B7       ;DRAW BOTTOM LINE
6083: A0 00              LDY  #$00
6085: BD 3B 67           LDA  HI,X
6088: 85 1B              STA  HIGH
608A: BD FB 67           LDA  LO,X
608D: 85 1A              STA  LOW
608F: A9 7F              LDA  #$7F
6091: 91 1A      LN      STA  (LOW),Y
6093: C8                 INY
6094: C0 27              CPY  #$27
6096: 90 F9              BLT  LN
6098: A2 0C              LDX  #$0C       ;DRAW TOP LINE
609A: A0 00              LDY  #$00
609C: BD 3B 67           LDA  HI,X
609F: 85 1B              STA  HIGH
60A1: BD FB 67           LDA  LO,X
60A4: 85 1A              STA  LOW
60A6: A9 7F              LDA  #$7F
60A8: 91 1A      LN1     STA  (LOW),Y
60AA: C8                 INY
60AB: C0 14              CPY  #$14
60AD: 90 F9              BLT  LN1
                 ********** MAIN PROGRAM **********
60AF: 20 03 61           JSR  MINITIAL   ;SET LINE & DEPTH OF MAN
60B2: 20 12 61           JSR  BINITIAL   ;SET LINE FOR BULLET
60B5: 20 1D 61           JSR  SINITIAL   ;ZERO SCORE COUNTER
60B8: 20 31 61   PADDLE  JSR  PDLE       ;READ PADDLE
60BB: 20 6C 61           JSR  MDRAW      ;DRAW MAN
60BE: AD 0B 60           LDA  BULON
60C1: C9 01              CMP  #$01       ;IS BULLET ON?
60C3: F0 16              BEQ  BULLET     ;IF YES, CONTINUE BULLET DRAW
60C5: AD 61 C0           LDA  BUTTON     ;IF NO, IS BUTTON PRESSED?
60C8: 30 0C              BMI  BULLET1    ;IF YES, DRAW BULLET
60CA: AD 0D 60           LDA  DELAY      ;IF NO,
60CD: 20 A8 FC           JSR  WAIT           DELAY AND
60D0: 20 6C 61           JSR  MDRAW          ERASE MAN AND
60D3: 4C B8 60           JMP  PADDLE         READ PADDLE AGAIN
60D6: A9 01      BULLET1 LDA  #$01       ;SET BULLET ON
60D8: 8D 0B 60           STA  BULON
60DB: 20 B6 61   BULLET  JSR  LOADBUL    ;LOAD BULLET SHAPE INTO BTEMP
60DE: 20 D9 61           JSR  BDRAW      ;DRAW BULLET & TEST FOR COLLISION
60E1: AD 0D 60           LDA  DELAY
60E4: 20 A8 FC           JSR  WAIT       ;DELAY
60E7: 20 0F 62           JSR  BXDRAW     ;ERASE BULLET
60EA: 20 6C 61           JSR  MDRAW      ;ERASE MAN
60ED: AD 05 60           LDA  BLINE
60F0: 38                 SEC
60F1: E9 08              SBC  #$08       ;MOVE BLINE UP 8 LINES
60F3: 8D 05 60           STA  BLINE
60F6: C9 05              CMP  #$05       ;LESS THAN 5 LINES FROM TOP?
60F8: 90 03              BLT  TOP        ;IF YES TAKE BRANCH
60FA: 4C B8 60           JMP  PADDLE     ;IF NO, READ PADDLE AGAIN
60FD: 20 12 61   TOP     JSR  BINITIAL   ;INITIALIZE BULLET LINE
6100: 4C B8 60           JMP  PADDLE     ;READ PADDLE
                 ********** SUBROUTINES **********
6103: A9 AA      MINITIAL LDA #$AA
```

```
6105: 8D 03 60           STA  MLINE
6108: 8D 04 60           STA  MLINEA
610B: 18                 CLC
610C: 69 0D              ADC  #$0D
610E: 8D 06 60           STA  DEPTH
6111: 60                 RTS
            ***********************
6112: A9 00   BINITIAL LDA  #$00        ;BULON = 0 IF
6114: 8D 0B 60           STA  BULON         BULLET NOT ON SCREEN
6117: A9 A4              LDA  #$A4
6119: 8D 05 60           STA  BLINE
611C: 60                 RTS
            **********************
611D: A9 00   SINITIAL LDA  #$00        ;SCORE DISPLAYS THREE 0'S
611F: 8D 39 60           STA  SUM
6122: 8D 3A 60           STA  COUNTER
6125: AA                 TAX
6126: A0 11              LDY  #$11
6128: 20 72 63  PR       JSR  PRINT
612B: C8                 INY
612C: C0 14              CPY  #$14
612E: 90 F8              BLT  PR
6130: 60                 RTS
            **********************
6131: A2 00     PDLE     LDX  #$00
6133: 20 1E FB           JSR  PREAD     ;READ PADDLE 0
6136: 98                 TYA
6137: 8D 07 60           STA  MHORIZ    ;0-255 IN MHORIZ
613A: AD 0B 60           LDA  BULON
613D: C9 01              CMP  #$01      ;IS BULLET ON?
613F: F0 06              BEQ  PDLE1     ;IF YES, TAKE BRANCH
6141: AD 07 60           LDA  MHORIZ    ;IF NO, SET BHORIZ EQUAL
6144: 8D 08 60           STA  BHORIZ        TO MHORIZ
6147: AC 07 60  PDLE1    LDY  MHORIZ
614A: B9 35 65           LDA  BYTETBL,Y ;CONVERT 0-255 TO 0-36 (BYTE)
614D: 8D 0A 60           STA  HORIZM    ;MAN BYTE POSITION
6150: B9 38 66           LDA  OFFSET,Y  ;GET SHAPE NUMBER
6153: 0A                 ASL            ;LOAD SHAPE INTO MTEMP
6154: AA                 TAX
6155: BD 3B 60           LDA  MSHPADR,X
6158: 85 1A              STA  LOW
615A: BD 3C 60           LDA  MSHPADR+1,X
615D: 85 1B              STA  HIGH
615F: A0 00              LDY  #$00
6161: B1 1A     LOAD     LDA  (LOW),Y
6163: 99 0F 60           STA  MTEMP,Y
6166: C8                 INY
6167: C0 27              CPY  #$27
6169: 90 F6              BLT  LOAD
616B: 60                 RTS
            ***********************
616C: A9 00     MDRAW    LDA  #$00
616E: 8D 0C 60           STA  XCOUNT
6171: AE 03 60  MDRAW1   LDX  MLINE
6174: AC 0A 60           LDY  HORIZM
6177: BD 3B 67           LDA  HI,X
617A: 85 1B              STA  HIGH
617C: BD FB 67           LDA  LO,X
617F: 85 1A              STA  LOW
6181: AE 0C 60           LDX  XCOUNT
6184: B1 1A              LDA  (LOW),Y
```

```
6186: 5D 0F 60         EOR  MTEMP,X
6189: 91 1A            STA  (LOW),Y
618B: C8               INY
618C: B1 1A            LDA  (LOW),Y
618E: 5D 10 60         EOR  MTEMP+1,X
6191: 91 1A            STA  (LOW),Y
6193: C8               INY
6194: B1 1A            LDA  (LOW),Y
6196: 5D 11 60         EOR  MTEMP+2,X
6199: 91 1A            STA  (LOW),Y
619B: EE 0C 60         INC  XCOUNT
619E: EE 0C 60         INC  XCOUNT
61A1: EE 0C 60         INC  XCOUNT
61A4: EE 03 60         INC  MLINE
61A7: AD 03 60         LDA  MLINE
61AA: CD 06 60         CMP  DEPTH
61AD: 90 C2            BLT  MDRAW1
61AF: AD 04 60         LDA  MLINEA      ;RESET LINE
61B2: 8D 03 60         STA  MLINE
61B5: 60               RTS
                 ***********************
61B6: AC 08 60   LOADBUL LDY  BHORIZ      ;CONVERTS 0-255 TO
61B9: B9 35 65         LDA  BYTETBL,Y      SCREEN BYTE (0-36)
61BC: 18               CLC              ;ADD 2 TO ALIGN BULLET
61BD: 69 02            ADC  #$02           WITH GUN
61BF: 8D 09 60         STA  HORIZB      ;BULLET BYTE POSITION
61C2: B9 38 66         LDA  OFFSET,Y    ;GET BULLET SHAPE NUMBER
61C5: 0A               ASL              ;LOAD BULLET SHAPE INTO BTEMP
61C6: AA               TAX
61C7: BD 49 60         LDA  BSHPADR,X
61CA: 85 1A            STA  LOW
61CC: BD 4A 60         LDA  BSHPADR+1,X
61CF: 85 1B            STA  HIGH
61D1: A0 00            LDY  #$00
61D3: B1 1A            LDA  (LOW),Y
61D5: 8D 0E 60         STA  BTEMP
61D8: 60               RTS
                 ***********************
61D9: AE 05 60   BDRAW  LDX  BLINE
61DC: AC 09 60         LDY  HORIZB
61DF: BD 3B 67         LDA  HI,X
61E2: 85 1B            STA  HIGH
61E4: BD FB 67         LDA  LO,X
61E7: 85 1A            STA  LOW
61E9: B1 1A            LDA  (LOW),Y
61EB: 2D 0E 60         AND  BTEMP       ;RESULT IS 0 IF NO COLLISION
61EE: C9 00            CMP  #$00
61F0: F0 03            BEQ  NOHIT
61F2: 4C FD 61         JMP  COLLISION
61F5: B1 1A      NOHIT  LDA  (LOW),Y    ;DRAW BULLET
61F7: 4D 0E 60         EOR  BTEMP
61FA: 91 1A            STA  (LOW),Y
61FC: 60               RTS
                 ************************
61FD: 20 27 62   COLLISION JSR EXPLODE
6200: EE 39 60         INC  SUM         ;ADD 1 TO SCORE
6203: 20 28 63         JSR  SCORE       ;DISPLAY SCORE
6206: 20 12 61         JSR  BINITIAL
6209: 20 6C 61         JSR  MDRAW       ;ERASE MAN
620C: 4C B8 60         JMP  PADDLE
                 ************************
```

```
620F: AE 05 60   BXDRAW   LDX  BLINE      ;BDRAW WITHOUT COLLISION TEST
6212: AC 09 60            LDY  HORIZB
6215: BD 3B 67            LDA  HI,X
6218: 85 1B               STA  HIGH
621A: BD FB 67            LDA  LO,X
621D: 85 1A               STA  LOW
621F: B1 1A               LDA  (LOW),Y
6221: 4D 0E 60            EOR  BTEMP
6224: 91 1A               STA  (LOW),Y
6226: 60                  RTS
                 ***********************
6227: 20 D8 62   EXPLODE  JSR  INITE1
622A: 20 6C 62            JSR  DRAWE1     ;DRAW
622D: A9 60               LDA  #$60
622F: 20 A8 FC            JSR  WAIT
6232: 20 D8 62            JSR  INITE1
6235: 20 6C 62            JSR  DRAWE1     ;ERASE
6238: 20 EC 62            JSR  INITE2
623B: 20 6C 62            JSR  DRAWE1     ;DRAW
623E: A9 BB               LDA  #$BB
6240: 20 A8 FC            JSR  WAIT
6243: 20 EC 62            JSR  INITE2
6246: 20 6C 62            JSR  DRAWE1     ;ERASE
6249: 20 00 63            JSR  INITE3
624C: 20 6C 62            JSR  DRAWE1     ;DRAW
624F: A9 BB               LDA  #$BB
6251: 20 A8 FC            JSR  WAIT
6254: 20 00 63            JSR  INITE3
6257: 20 6C 62            JSR  DRAWE1     ;ERASE
625A: 20 14 63            JSR  INITE4
625D: 20 9B 62            JSR  DRAWE2     ;DRAW
6260: A9 FF               LDA  #$FF
6262: 20 A8 FC            JSR  WAIT
6265: 20 14 63            JSR  INITE4
6268: 20 9B 62            JSR  DRAWE2     ;ERASE
626B: 60                  RTS
                 *************************
626C: AC 09 60   DRAWE1   LDY  HORIZB     ;ROUTINE FOR FIRST 3
626F: AE 36 60            LDX  ELINE           EXPLOSION SHAPES
6272: BD 3B 67            LDA  HI,X
6275: 85 1B               STA  HIGH
6277: BD FB 67            LDA  LO,X
627A: 85 1A               STA  LOW
627C: AE 0C 60            LDX  XCOUNT
627F: B1 1A               LDA  (LOW),Y
6281: 5D 0B 65            EOR  ESHAPE,X
6284: 91 1A               STA  (LOW),Y
6286: EE 0C 60            INC  XCOUNT
6289: EE 36 60            INC  ELINE
628C: AD 36 60            LDA  ELINE
628F: CD 38 60            CMP  EDEPTH
6292: 90 D8               BLT  DRAWE1
6294: AD 37 60            LDA  ELINEA
6297: 8D 36 60            STA  ELINE
629A: 60                  RTS
                 ***********************
629B: AC 09 60   DRAWE2   LDY  HORIZB     ;ROUTINE FOR FOURTH
629E: AE 36 60            LDX  ELINE           EXPLOSION SHAPE
62A1: BD 3B 67            LDA  HI,X
62A4: 85 1B               STA  HIGH
62A6: BD FB 67            LDA  LO,X
```

```
62A9: 85 1A             STA  LOW
62AB: AE 0C 60          LDX  XCOUNT
62AE: B1 1A             LDA  (LOW),Y
62B0: 5D 0B 65          EOR  ESHAPE,X
62B3: 91 1A             STA  (LOW),Y
62B5: EE 0C 60          INC  XCOUNT
62B8: C8                INY
62B9: AE 0C 60          LDX  XCOUNT
62BC: B1 1A             LDA  (LOW),Y
62BE: 5D 0B 65          EOR  ESHAPE,X
62C1: 91 1A             STA  (LOW),Y
62C3: EE 0C 60          INC  XCOUNT
62C6: EE 36 60          INC  ELINE
62C9: AD 36 60          LDA  ELINE
62CC: CD 38 60          CMP  EDEPTH
62CF: 90 CA             BLT  DRAWE2
62D1: AD 37 60          LDA  ELINEA
62D4: 8D 36 60          STA  ELINE
62D7: 60                RTS
                 ************************
62D8: A9 00      INITE1 LDA  #$00      ;INITIALIZE FIRST EXPLOSION
62DA: 8D 0C 60          STA  XCOUNT
62DD: A9 09             LDA  #$09
62DF: 8D 37 60          STA  ELINEA
62E2: 8D 36 60          STA  ELINE
62E5: 18                CLC
62E6: 69 05             ADC  #$05
62E8: 8D 38 60          STA  EDEPTH
62EB: 60                RTS
62EC: A9 05      INITE2 LDA  #$05      ;INITIALIZE SECOND EXPLOSION
62EE: 8D 0C 60          STA  XCOUNT
62F1: A9 09             LDA  #$09
62F3: 8D 37 60          STA  ELINEA
62F6: 8D 36 60          STA  ELINE
62F9: 18                CLC
62FA: 69 05             ADC  #$05
62FC: 8D 38 60          STA  EDEPTH
62FF: 60                RTS
6300: A9 0A      INITE3 LDA  #$0A      ;INITIALIZE THIRD EXPLOSION
6302: 8D 0C 60          STA  XCOUNT
6305: A9 05             LDA  #$05
6307: 8D 37 60          STA  ELINEA
630A: 8D 36 60          STA  ELINE
630D: 18                CLC
630E: 69 08             ADC  #$08
6310: 8D 38 60          STA  EDEPTH
6313: 60                RTS
6314: A9 12      INITE4 LDA  #$12      ;INITIALIZE FOURTH EXPLOSION
6316: 8D 0C 60          STA  XCOUNT
6319: A9 01             LDA  #$01
631B: 8D 37 60          STA  ELINEA
631E: 8D 36 60          STA  ELINE
6321: 18                CLC
6322: 69 0C             ADC  #$0C
6324: 8D 38 60          STA  EDEPTH
6327: 60                RTS
                 ************************
6328: AD 39 60   SCORE  LDA  SUM       ;GET SCORE (0-9)
632B: C9 0A             CMP  #$0A      ;EQUAL TO 10?
632D: F0 0A             BEQ  C10       ;IF YES, BRANCH
632F: 0A                ASL            ;IF NO, MULTIPLY BY 8
```

```
6330: 0A                ASL
6331: 0A                ASL
6332: AA                TAX
6333: A0 13             LDY   #$13        ;BYTE POSITION FOR FIRST DIGIT
6335: 20 72 63          JSR   PRINT       ;PRINT DIGIT
6338: 60                RTS
6339: EE 3A 60   C10    INC   COUNTER     ;INC COUNTER (INITIALLY 0)
633C: AD 3A 60          LDA   COUNTER
633F: C9 0A             CMP   #$0A        ;EQUAL TO 10?
6341: F0 11             BEQ   STOP1       ;IF YES, PRINT 100 AND STOP GAME
6343: 0A                ASL               ;IF NO, MULTIPLY BY 8
6344: 0A                ASL
6345: 0A                ASL
6346: AA                TAX
6347: A0 12             LDY   #$12        ;BYTE POSITION OF MIDDLE DIGIT
6349: 20 72 63          JSR   PRINT       ;PRINT DIGIT
634C: A9 00             LDA   #$00        ;ZERO SUM AND
634E: 8D 39 60          STA   SUM           RETURN TO PRINT 0
6351: 4C 28 63          JMP   SCORE           IN FIRST DIGIT POSITION
6354: A2 08      STOP1  LDX   #$08        ;ACCESSES DIGIT "1"
6356: A0 11             LDY   #$11        ;BYTE POSITION OF LEFTMOST DIGIT
6358: 20 72 63          JSR   PRINT       ;PRINT DIGIT
635B: A2 00             LDX   #$00        ;ACCESSES DIGIT "0"
635D: A0 12             LDY   #$12        ;BYTE POSITION OF MIDDLE DIGIT
635F: 20 72 63          JSR   PRINT       ;PRINT DIGIT
6362: A0 13             LDY   #$13        ;BYTE POSITION OF FIRST DIGIT
6364: 20 72 63          JSR   PRINT       ;PRINT DIGIT
6367: 2C 00 C0   STOP   BIT   $C000       ;ANY KEY PRESSED?
636A: 10 FB             BPL   STOP        ;IF NO, BRANCH BACK & WAIT
636C: 2C 10 C0          BIT   $C010       ;IF YES, CLEAR KEYBOARD STROBE AND
636F: 4C 57 60          JMP   PGM           START PROGRAM OVER
                 *************************
6372: BD A3 63   PRINT  LDA   NSHAPE,X    ;RETRIEVE NUMBER SHAPE
6375: 99 D0 23          STA   $23D0,Y     ;LINE #$B8 (184)
6378: BD A4 63          LDA   NSHAPE+1,X
637B: 99 D0 27          STA   $27D0,Y     ;LINE #$B9 (185)
637E: BD A5 63          LDA   NSHAPE+2,X
6381: 99 D0 2B          STA   $2BD0,Y     ;LINE #$BA (186)
6384: BD A6 63          LDA   NSHAPE+3,X
6387: 99 D0 2F          STA   $2FD0,Y     ;LINE #$BB (187)
638A: BD A7 63          LDA   NSHAPE+4,X
638D: 99 D0 33          STA   $33D0,Y     ;LINE #$BC (188)
6390: BD A8 63          LDA   NSHAPE+5,X
6393: 99 D0 37          STA   $37D0,Y     ;LINE #$BD (189)
6396: BD A9 63          LDA   NSHAPE+6,X
6399: 99 D0 3B          STA   $3BD0,Y     ;LINE #$BE (190)
639C: BD AA 63          LDA   NSHAPE+7,X
639F: 99 D0 3F          STA   $3FD0,Y     ;LINE #$BF (191)
63A2: 60                RTS
                 ****************************
63A3: 00 1C 22   NSHAPE HEX   001C22222222221C    ;NUMBER SHAPES - "0"
63A6: 22 22 22 22 1C
63AB: 00 08 0C          HEX   00080C080808081C    ;"1"
63AE: 08 08 08 08 1C
63B3: 00 1C 22          HEX   001C22201008043E    ;"2"
63B6: 20 10 08 04 3E
63BB: 00 1C 22          HEX   001C22201C20221C    ;"3"
63BE: 20 1C 20 22 1C
63C3: 00 10 18          HEX   00101814123E1010    ;"4"
63C6: 14 12 3E 10 10
63CB: 00 3E 02          HEX   003E021E2020201E    ;"5"
```

```
63CE: 1E 20 20 20 1E
63D3: 00 1C 22  436          HEX  001C22021E22221C     ;"6"
63D6: 02 1E 22 22 1C
63DB: 00 3E 20  437          HEX  003E201008040404     ;"7"
63DE: 10 08 04 04 04
63E3: 00 1C 22  438          HEX  001C22221C22221C     ;"8"
63E6: 22 1C 22 22 1C
63EB: 00 1C 22  439          HEX  001C22223C20221C     ;"9"
63EE: 22 3C 20 22 1C
63F3: 00 0E 01  440  MSHAPE1 HEX  000E01000E01000E01   ;MAN SHAPE TABLES
63F6: 00 0E 01 00 0E 01
63FC: 00 44 01  441          HEX  004401007F00601F00
63FF: 00 7F 00 60 1F 00
6405: 30 1F 00  442          HEX  301F00181F00001F00
6408: 18 1F 00 00 1F 00
640E: 00 1F 00  443          HEX  001F00001B00403100
6411: 00 1B 00 40 31 00
6417: 60 60 00  444          HEX  606000
641A: 00 1C 02  445  MSHAPE2 HEX  001C02001C02001C02
641D: 00 1C 02 00 1C 02
6423: 00 08 03  446          HEX  000803007E01003E00
6426: 00 7E 01 00 3E 00
642C: 00 3F 00  447          HEX  003F00403F00003E00
642F: 40 3F 00 00 3E 00
6435: 00 3E 00  448          HEX  003E00003600003600
6438: 00 36 00 00 36 00
643E: 00 63 00  449          HEX  006300
6441: 00 38 04  450  MSHAPE3 HEX  003804003804003804
6444: 00 38 04 00 38 04
644A: 00 10 06  451          HEX  001006007C03007C00
644D: 00 7C 03 00 7C 00
6453: 00 7C 00  452          HEX  007C00007E00007C00
6456: 00 7E 00 00 7C 00
645C: 00 38 00  453          HEX  003800003800006C00
645F: 00 38 00 00 6C 00
6465: 00 46 01  454          HEX  004601
6468: 00 70 08  455  MSHAPE4 HEX  007008007008007008
646B: 00 70 08 00 70 08
6471: 00 20 0C  456          HEX  00200C007807007801
6474: 00 78 07 00 78 01
647A: 00 78 01  457          HEX  007801007801007801
647D: 00 78 01 00 78 01
6483: 00 70 00  458          HEX  007000007000007000
6486: 00 70 00 00 70 00
648C: 00 70 00  459          HEX  007000
648F: 00 60 11  460  MSHAPE5 HEX  006011006011006011
6492: 00 60 11 00 60 11
6498: 00 40 18  461          HEX  00401800700F007003
649B: 00 70 0F 00 70 03
64A1: 00 70 03  462          HEX  007003007803007003
64A4: 00 78 03 00 70 03
64AA: 00 60 01  463          HEX  006001006001003003
64AD: 00 60 01 00 30 03
64B3: 00 18 06  464          HEX  001806
64B6: 00 40 23  465  MSHAPE6 HEX  004023004023004023
64B9: 00 40 23 00 40 23
64BF: 00 00 31  466          HEX  00003100601F006007
64C2: 00 60 1F 00 60 07
64C8: 00 70 07  467          HEX  007007007807006007
64CB: 00 78 07 00 60 07
64D1: 00 60 07  468          HEX  006007006006006006
```

```
64D4: 00 60 06 00 60 06
64DA: 00 30 0C  469            HEX  00300C
64DD: 00 00 47  470  MSHAPE7   HEX  000047000047000047
64E0: 00 00 47 00 00 47
64E6: 00 00 62  471            HEX  00006200403F00700F
64E9: 00 40 3F 00 70 0F
64EF: 00 58 0F  472            HEX  00580F004C0F00400F
64F2: 00 4C 0F 00 40 0F
64F8: 00 40 0F  473            HEX  00400F00400D006018
64FB: 00 40 0D 00 60 18
6501: 00 30 30  474            HEX  003030
6504: 01        475  BSHAPE1   HEX  01             ;BULLET SHAPES
6505: 02        476  BSHAPE2   HEX  02
6506: 04        477  BSHAPE3   HEX  04
6507: 08        478  BSHAPE4   HEX  08
6508: 10        479  BSHAPE5   HEX  10
6509: 20        480  BSHAPE6   HEX  20
650A: 40        481  BSHAPE7   HEX  40
650B: 28 22 1A  482  ESHAPE    HEX  28221A2514     ;EXPLOSION SHAPES  - NO. 1
650E: 25 14
6510: 2C 52 44  483            HEX  2C5244320C       ;NO. 2
6513: 32 0C
6515: 38 3E 7F  484            HEX  383E7F7E7E3F3F1C ;NO. 3
6518: 7E 7E 3F 3F 1C
651D: 18 06 7C  485            HEX  18067C0F7C3F7E3F ;NO. 4
6520: 0F 7C 3F 7E 3F
6525: 7C 7F 7C  486            HEX  7C7F7C3F7E3F7F1F
6528: 3F 7E 3F 7F 1F
652D: 7E 0F 7C  487            HEX  7E0F7C1F700F4003
6530: 1F 70 0F 40 03
                     BYTETBL
                     OFFSET
                     HI
                     LO
```

2235 bytes

Symbol table - numerical order:

LOW	=$1A	HIGH	=$1B	MLINE	=$6003	MLINEA	=$6004
BLINE	=$6005	DEPTH	=$6006	MHORIZ	=$6007	BHORIZ	=$6008
HORIZB	=$6009	HORIZM	=$600A	BULON	=$600B	XCOUNT	=$600C
DELAY	=$600D	BTEMP	=$600E	MTEMP	=$600F	ELINE	=$6036
ELINEA	=$6037	EDEPTH	=$6038	SUM	=$6039	COUNTER	=$603A
MSHPADR	=$603B	BSHPADR	=$6049	PGM	=$6057	CLR1	=$606B
CLR	=$606F	LN	=$6091	LN1	=$60A8	PADDLE	=$60B8
BULLET1	=$60D6	BULLET	=$60DB	TOP	=$60FD	MINITIAL	=$6103
BINITIAL	=$6112	SINITIAL	=$611D	PR	=$6128	PDLE	=$6131
PDLE1	=$6147	LOAD	=$6161	MDRAW	=$616C	MDRAW1	=$6171
LOADBUL	=$61B6	BDRAW	=$61D9	NOHIT	=$61F5	COLLISION	=$61FD
BXDRAW	=$620F	EXPLODE	=$6227	DRAWE1	=$626C	DRAWE2	=$629B
INITE1	=$62D8	INITE2	=$62EC	INITE3	=$6300	INITE4	=$6314
SCORE	=$6328	C10	=$6339	STOP1	=$6354	STOP	=$6367
PRINT	=$6372	NSHAPE	=$63A3	MSHAPE1	=$63F3	MSHAPE2	=$641A
MSHAPE3	=$6441	MSHAPE4	=$6468	MSHAPE5	=$648F	MSHAPE6	=$64B6
MSHAPE7	=$64DD	BSHAPE1	=$6504	BSHAPE2	=$6505	BSHAPE3	=$6506
BSHAPE4	=$6507	BSHAPE5	=$6508	BSHAPE6	=$6509	BSHAPE7	=$650A
ESHAPE	=$650B	BYTETBL	=$6535	OFFSET	=$6638	HI	=$673B
LO	=$67FB	GRAPHICS	=$C050	MIXOFF	=$C052	PAGE1	=$C054
HIRES	=$C057	BUTTON	=$C061	PREAD	=$FB1E	WAIT	=$FCA8

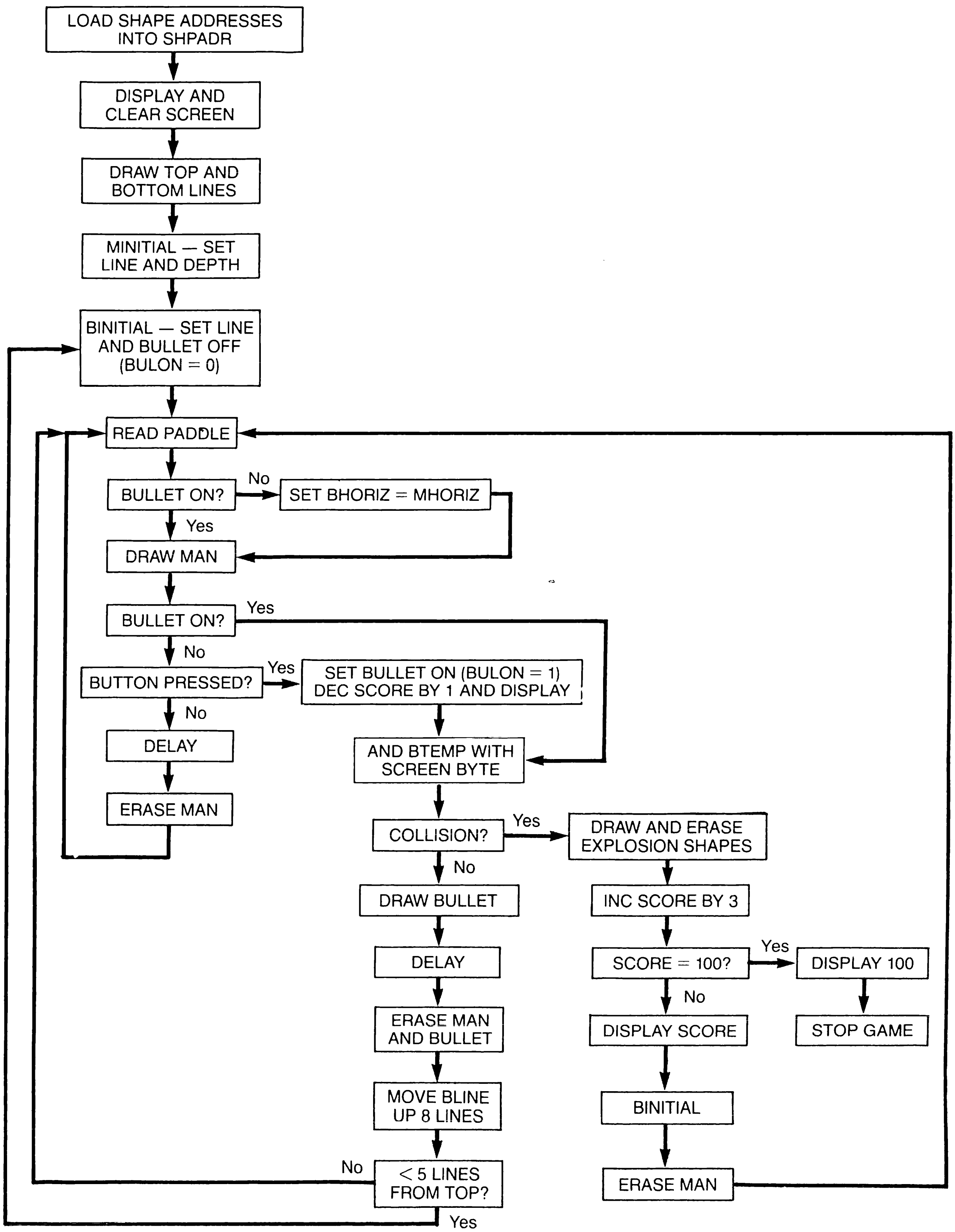
COUNTING BY MULTIPLES AND DECREMENTING SCORE

The counting routine in Program 8-1 fits in well with our game program where we increment the score by 1 every time a plane is hit, and stop the program when we reach 100 (or until 100 planes have appeared). However, it does have some limitations. First, if we want to increment in jumps greater than one, we have a problem. If we increment by 3 and go from 9 to 12, for example, the counter will display 010 because SUM is zeroed after we increment COUNTER. Second, if we want to decrement the score, say by 1 each time the bullet is fired, the routine will not handle this at the 9-0 boundaries (e.g., from 10 to 9), and there would also be a problem if the score is decremented while at 000. The following program (Program 8-2) presents a score routine that solves all these problems.

The SINITIAL and PRINT subroutines are the same in Program 8-2 as in Program 8-1 and again we use SUM as the 1's counter and COUNTER as the 10's counter. The differences are in the MAIN PROGRAM, COLLISION, and SCORE routines. In the MAIN PROGRAM, we decrement the score by one each time a bullet is fired and jump to SCORE to display the count (lines 119 and 120). In COLLISION, SUM is incremented by three each time a collision is detected (lines 254 to 256). Let's see how the SCORE routine handles these changes.

First, we load the Accumulator with SUM and compare to #$FF. If SUM initially contains zero, as it would if the count were 010, 020, 030, etc., and is then decremented by 1 before we jump to the SCORE routine, we want to change its value to 9 and then decrement COUNTER by 1. Decrementing #$00 by 1 results in a value of #$FF. Thus, SUM would contain #$FF and the branch at line 382 would not be taken. The program would continue to line 383 where #$09 is loaded into SUM and COUNTER is decremented by 1. We now want to see if the score was at 000 at the time SUM was decremented by a bullet firing—obviously at this point we don't want to decrement the score, but rather retain the 000 display. If the score is 000, then both SUM and COUNTER contain zero. Thus, decrementing COUNTER will yield a value of #$FF and the branch at line 388 will not be taken. The program would continue to line 389 where COUNTER and SUM are both set to zero in preparation for printing 000. These CMP #$FF instructions then are used for the special situations where either SUM = 0 and is decremented, or where both SUM and COUNTER = 0 and SUM is decremented.

If SUM doesn't contain zero before decrementing or is simply not decremented, SUM will not contain #$FF, the branch at line 382 is taken, and the program proceeds to C1 (line 392). Here SUM is compared to 10 and if less than 10, the program branches to C10 for the same print routine as in Program 8-1. If SUM equals or is greater than 10, the branch at line 394 is not taken, COUNTER is incremented by 1, 10 is subtracted from SUM, and the count is then printed. Thus, if SUM contains 12, subtracting 10 leaves 2 in SUM. This figure is then printed in the rightmost digit position, and COUNTER will contain 1, which is printed in the middle digit position, producing the display 012.


LOAD SHAPE ADDRESSES INTO SHPADR
DISPLAY AND CLEAR SCREEN
DRAW TOP AND BOTTOM LINES
MINITIAL — SET LINE AND DEPTH
BINITIAL — SET LINE AND BULLET OFF (BULON = 0)
READ PADDLE
BULLET ON?
No
SET BHORIZ = MHORIZ
Yes
DRAW MAN
BULLET ON?
Yes
No
BUTTON PRESSED?
Yes
SET BULLET ON (BULON = 1) DEC SCORE BY 1 AND DISPLAY
No
DELAY
AND BTEMP WITH SCREEN BYTE
ERASE MAN
COLLISION?
Yes
DRAW AND ERASE EXPLOSION SHAPES
No
DRAW BULLET
INC SCORE BY 3
DELAY
SCORE = 100?
Yes
DISPLAY 100
No
ERASE MAN AND BULLET
DISPLAY SCORE
STOP GAME
MOVE BLINE UP 8 LINES
BINITIAL
No
< 5 LINES FROM TOP?
ERASE MAN
Yes

```
]PROGRAM 8-2
:ASM
                *COLLISION AND EXPLOSION WITH SCORING*DECREMENTING SCORE
                         ORG  $6000
6000: 4C 57 60           JMP  PGM
                MLINE    DS   1
                MLINEA   DS   1
                BLINE    DS   1
                DEPTH    DS   1
                MHORIZ   DS   1
                BHORIZ   DS   1
                HORIZB   DS   1
                HORIZM   DS   1
                BULON    DS   1
                XCOUNT   DS   1
                DELAY    DS   1
                BTEMP    DS   1
                MTEMP    DS   39
                ELINE    DS   1
                ELINEA   DS   1
                EDEPTH   DS   1
                SUM      DS   1
                COUNTER  DS   1
                GRAPHICS =    $C050
                MIXOFF   =    $C052
                HIRES    =    $C057
                PAGE1    =    $C054
                HIGH     =    $1B
                LOW      =    $1A
                WAIT     =    $FCA8
                PREAD    =    $FB1E
                BUTTON   =    $C061      ;BUTTON 0
                *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                *CONTINUE FOR ALL 7 SHAPES
603B: 1E        MSHPADR  DFB  #<MSHAPE1
603C: 64                 DFB  #>MSHAPE1
603D: 45                 DFB  #<MSHAPE2
603E: 64                 DFB  #>MSHAPE2
603F: 6C                 DFB  #<MSHAPE3
6040: 64                 DFB  #>MSHAPE3
6041: 93                 DFB  #<MSHAPE4
6042: 64                 DFB  #>MSHAPE4
6043: BA                 DFB  #<MSHAPE5
6044: 64                 DFB  #>MSHAPE5
6045: E1                 DFB  #<MSHAPE6
6046: 64                 DFB  #>MSHAPE6
6047: 08                 DFB  #<MSHAPE7
6048: 65                 DFB  #>MSHAPE7
6049: 2F        BSHPADR  DFB  #<BSHAPE1
604A: 65                 DFB  #>BSHAPE1
604B: 30                 DFB  #<BSHAPE2
604C: 65                 DFB  #>BSHAPE2
604D: 31                 DFB  #<BSHAPE3
604E: 65                 DFB  #>BSHAPE3
604F: 32                 DFB  #<BSHAPE4
6050: 65                 DFB  #>BSHAPE4
6051: 33                 DFB  #<BSHAPE5
6052: 65                 DFB  #>BSHAPE5
6053: 34                 DFB  #<BSHAPE6
```

```
6054: 65               DFB  #>BSHAPE6
6055: 35               DFB  #<BSHAPE7
6056: 65               DFB  #>BSHAPE7
6057: AD 50 C0  PGM    LDA  GRAPHICS   ;HIRES,P.1
605A: AD 52 C0         LDA  MIXOFF
605D: AD 57 C0         LDA  HIRES
6060: AD 54 C0         LDA  PAGE1
6063: A9 00            LDA  #$00       ;CLEAR SCREEN 1
6065: 85 1A            STA  LOW
6067: A9 20            LDA  #$20
6069: 85 1B            STA  HIGH
606B: A0 00     CLR1   LDY  #$00
606D: A9 00            LDA  #$00
606F: 91 1A     CLR    STA  (LOW),Y
6071: C8               INY
6072: D0 FB            BNE  CLR
6074: E6 1B            INC  HIGH
6076: A5 1B            LDA  HIGH
6078: C9 40            CMP  #$40
607A: 90 EF            BLT  CLR1
607C: A9 50            LDA  #$50       ;LOAD DELAY
607E: 8D 0D 60         STA  DELAY
6081: A2 B7            LDX  #$B7       ;DRAW BOTTOM LINE
6083: A0 00            LDY  #$00
6085: BD 66 67         LDA  HI,X
6088: 85 1B            STA  HIGH
608A: BD 26 68         LDA  LO,X
608D: 85 1A            STA  LOW
608F: A9 7F            LDA  #$7F
6091: 91 1A     LN     STA  (LOW),Y
6093: C8               INY
6094: C0 27            CPY  #$27
6096: 90 F9            BLT  LN
6098: A2 0C            LDX  #$0C       ;DRAW TOP LINE
609A: A0 00            LDY  #$00
609C: BD 66 67         LDA  HI,X
609F: 85 1B            STA  HIGH
60A1: BD 26 68         LDA  LO,X
60A4: 85 1A            STA  LOW
60A6: A9 7F            LDA  #$7F
60A8: 91 1A     LN1    STA  (LOW),Y
60AA: C8               INY
60AB: C0 14            CPY  #$14
60AD: 90 F9            BLT  LN1
                ********** MAIN PROGRAM **********
60AF: 20 09 61         JSR  MINITIAL   ;SET LINE & DEPTH OF MAN
60B2: 20 18 61         JSR  BINITIAL   ;SET LINE FOR BULLET
60B5: 20 23 61         JSR  SINITIAL   ;ZERO SCORE COUNTER
60B8: 20 37 61  PADDLE JSR  PDLE       ;READ PADDLE
60BB: 20 72 61         JSR  MDRAW      ;DRAW MAN
60BE: AD 0B 60         LDA  BULON
60C1: C9 01            CMP  #$01       ;IS BULLET ON?
60C3: F0 1C            BEQ  BULLET     ;IF YES, CONTINUE BULLET DRAW
60C5: AD 61 C0         LDA  BUTTON     ;IF NO, IS BUTTON PRESSED?
60C8: 30 0C            BMI  BULLET1    ;IF YES, DRAW BULLET
60CA: AD 0D 60         LDA  DELAY      ;IF NO,
60CD: 20 A8 FC         JSR  WAIT           DELAY AND
60D0: 20 72 61         JSR  MDRAW          ERASE MAN AND
```

```
60D3: 4C B8 60             JMP  PADDLE          READ PADDLE AGAIN
60D6: A9 01       BULLET1  LDA  #$01        ;SET BULLET ON
60D8: 8D 0B 60             STA  BULON
60DB: CE 39 60             DEC  SUM         ;DECREMENT SUM AND
60DE: 20 34 63             JSR  SCORE           DISPLAY SCORE
60E1: 20 BC 61    BULLET   JSR  LOADBUL     ;LOAD BULLET SHAPE INTO BTEMP
60E4: 20 DF 61             JSR  BDRAW       ;DRAW BULLET & TEST FOR COLLISION
60E7: AD 0D 60             LDA  DELAY
60EA: 20 A8 FC             JSR  WAIT        ;DELAY
60ED: 20 1B 62             JSR  BXDRAW      ;ERASE BULLET
60F0: 20 72 61             JSR  MDRAW       ;ERASE MAN
60F3: AD 05 60             LDA  BLINE
60F6: 38                   SEC
60F7: E9 08                SBC  #$08        ;MOVE BLINE UP 8 LINES
60F9: 8D 05 60             STA  BLINE
60FC: C9 05                CMP  #$05        ;LESS THAN 5 LINES FROM TOP?
60FE: 90 03                BLT  TOP         ;IF YES TAKE BRANCH
6100: 4C B8 60             JMP  PADDLE      ;IF NO, READ PADDLE AGAIN
6103: 20 18 61    TOP      JSR  BINITIAL    ;INITIALIZE BULLET LINE
6106: 4C B8 60             JMP  PADDLE      ;READ PADDLE
                  ********** SUBROUTINES **********
6109: A9 AA       MINITIAL LDA  #$AA
610B: 8D 03 60             STA  MLINE
610E: 8D 04 60             STA  MLINEA
6111: 18                   CLC
6112: 69 0D                ADC  #$0D
6114: 8D 06 60             STA  DEPTH
6117: 60                   RTS
                  ***********************
6118: A9 00       BINITIAL LDA  #$00        ;BULON = 0 IF
611A: 8D 0B 60             STA  BULON           BULLET NOT ON SCREEN
611D: A9 A4                LDA  #$A4
611F: 8D 05 60             STA  BLINE
6122: 60                   RTS
                  **********************
6123: A9 00       SINITIAL LDA  #$00        ;SCORE DISPLAYS THREE 0'S
6125: 8D 39 60             STA  SUM
6128: 8D 3A 60             STA  COUNTER
612B: AA                   TAX
612C: A0 11                LDY  #$11
612E: 20 9D 63    PR       JSR  PRINT
6131: C8                   INY
6132: C0 14                CPY  #$14
6134: 90 F8                BLT  PR
6136: 60                   RTS
                  **********************
6137: A2 00       PDLE     LDX  #$00
6139: 20 1E FB             JSR  PREAD       ;READ PADDLE 0
613C: 98                   TYA
613D: 8D 07 60             STA  MHORIZ      ;0-255 IN MHORIZ
6140: AD 0B 60             LDA  BULON
6143: C9 01                CMP  #$01        ;IS BULLET ON?
6145: F0 06                BEQ  PDLE1       ;IF YES, TAKE BRANCH
6147: AD 07 60             LDA  MHORIZ      ;IF NO, SET BHORIZ EQUAL
614A: 8D 08 60             STA  BHORIZ          TO MHORIZ
614D: AC 07 60    PDLE1    LDY  MHORIZ
6150: B9 60 65             LDA  BYTETBL,Y   ;CONVERT 0-255 TO 0-36 (BYTE)
6153: 8D 0A 60             STA  HORIZM      ;MAN BYTE POSITION
6156: B9 63 66             LDA  OFFSET,Y    ;GET SHAPE NUMBER
```

```
6159: 0A        175             ASL                ;LOAD SHAPE INTO MTEMP
615A: AA        176             TAX
615B: BD 3B 60  177             LDA  MSHPADR,X
615E: 85 1A     178             STA  LOW
6160: BD 3C 60  179             LDA  MSHPADR+1,X
6163: 85 1B     180             STA  HIGH
6165: A0 00     181             LDY  #$00
6167: B1 1A     182   LOAD      LDA  (LOW),Y
6169: 99 0F 60  183             STA  MTEMP,Y
616C: C8        184             INY
616D: C0 27     185             CPY  #$27
616F: 90 F6     186             BLT  LOAD
6171: 60        187             RTS
                188   ***********************
6172: A9 00     189   MDRAW     LDA  #$00
6174: 8D 0C 60  190             STA  XCOUNT
6177: AE 03 60  191   MDRAW1    LDX  MLINE
617A: AC 0A 60  192             LDY  HORIZM
617D: BD 66 67  193             LDA  HI,X
6180: 85 1B     194             STA  HIGH
6182: BD 26 68  195             LDA  LO,X
6185: 85 1A     196             STA  LOW
6187: AE 0C 60  197             LDX  XCOUNT
618A: B1 1A     198             LDA  (LOW),Y
618C: 5D 0F 60  199             EOR  MTEMP,X
618F: 91 1A     200             STA  (LOW),Y
6191: C8        201             INY
6192: B1 1A     202             LDA  (LOW),Y
6194: 5D 10 60  203             EOR  MTEMP+1,X
6197: 91 1A     204             STA  (LOW),Y
6199: C8        205             INY
619A: B1 1A     206             LDA  (LOW),Y
619C: 5D 11 60  207             EOR  MTEMP+2,X
619F: 91 1A     208             STA  (LOW),Y
61A1: EE 0C 60  209             INC  XCOUNT
61A4: EE 0C 60  210             INC  XCOUNT
61A7: EE 0C 60  211             INC  XCOUNT
61AA: EE 03 60  212             INC  MLINE
61AD: AD 03 60  213             LDA  MLINE
61B0: CD 06 60  214             CMP  DEPTH
61B3: 90 C2     215             BLT  MDRAW1
61B5: AD 04 60  216             LDA  MLINEA        ;RESET LINE
61B8: 8D 03 60  217             STA  MLINE
61BB: 60        218             RTS
                219   ***********************
61BC: AC 08 60  220   LOADBUL   LDY  BHORIZ        ;CONVERTS 0-255 TO
61BF: B9 60 65  221             LDA  BYTETBL,Y         SCREEN BYTE (0-36)
61C2: 18        222             CLC                ;ADD 2 TO ALIGN BULLET
61C3: 69 02     223             ADC  #$02              WITH GUN
61C5: 8D 09 60  224             STA  HORIZB        ;BULLET BYTE POSITION
61C8: B9 63 66  225             LDA  OFFSET,Y      ;GET BULLET SHAPE NUMBER
61CB: 0A        226             ASL                ;LOAD BULLET SHAPE INTO BTEMP
61CC: AA        227             TAX
61CD: BD 49 60  228             LDA  BSHPADR,X
61D0: 85 1A     229             STA  LOW
61D2: BD 4A 60  230             LDA  BSHPADR+1,X
61D5: 85 1B     231             STA  HIGH
61D7: A0 00     232             LDY  #$00
61D9: B1 1A     233             LDA  (LOW),Y
```

```
61DB: 8D 0E 60              STA   BTEMP
61DE: 60                    RTS
                  ************************
61DF: AE 05 60    BDRAW     LDX   BLINE
61E2: AC 09 60              LDY   HORIZB
61E5: BD 66 67              LDA   HI,X
61E8: 85 1B                 STA   HIGH
61EA: BD 26 68              LDA   LO,X
61ED: 85 1A                 STA   LOW
61EF: B1 1A                 LDA   (LOW),Y
61F1: 2D 0E 60              AND   BTEMP       ;RESULT IS 0 IF NO COLLISION
61F4: C9 00                 CMP   #$00
61F6: F0 03                 BEQ   NOHIT
61F8: 4C 03 62              JMP   COLLISION
61FB: B1 1A       NOHIT     LDA   (LOW),Y     ;DRAW BULLET
61FD: 4D 0E 60              EOR   BTEMP
6200: 91 1A                 STA   (LOW),Y
6202: 60                    RTS
                  *************************
6203: 20 33 62    COLLISION JSR EXPLODE
6206: EE 39 60              INC   SUM         ;ADD 1 TO SCORE
6209: EE 39 60              INC   SUM
620C: EE 39 60              INC   SUM
620F: 20 34 63              JSR   SCORE       ;DISPLAY SCORE
6212: 20 18 61              JSR   BINITIAL
6215: 20 72 61              JSR   MDRAW       ;ERASE MAN
6218: 4C B8 60              JMP   PADDLE
                  *************************
621B: AE 05 60    BXDRAW    LDX   BLINE       ;BDRAW WITHOUT COLLISION TEST
621E: AC 09 60              LDY   HORIZB
6221: BD 66 67              LDA   HI,X
6224: 85 1B                 STA   HIGH
6226: BD 26 68              LDA   LO,X
6229: 85 1A                 STA   LOW
622B: B1 1A                 LDA   (LOW),Y
622D: 4D 0E 60              EOR   BTEMP
6230: 91 1A                 STA   (LOW),Y
6232: 60                    RTS
                  ************************
6233: 20 E4 62    EXPLODE   JSR   INITE1
6236: 20 78 62              JSR   DRAWE1      ;DRAW
6239: A9 60                 LDA   #$60
623B: 20 A8 FC              JSR   WAIT
623E: 20 E4 62              JSR   INITE1
6241: 20 78 62              JSR   DRAWE1      ;ERASE
6244: 20 F8 62              JSR   INITE2
6247: 20 78 62              JSR   DRAWE1      ;DRAW
624A: A9 BB                 LDA   #$BB
624C: 20 A8 FC              JSR   WAIT
624F: 20 F8 62              JSR   INITE2
6252: 20 78 62              JSR   DRAWE1      ;ERASE
6255: 20 0C 63              JSR   INITE3
6258: 20 78 62              JSR   DRAWE1      ;DRAW
625B: A9 BB                 LDA   #$BB
625D: 20 A8 FC              JSR   WAIT
6260: 20 0C 63              JSR   INITE3
6263: 20 78 62              JSR   DRAWE1      ;ERASE
6266: 20 20 63              JSR   INITE4
6269: 20 A7 62              JSR   DRAWE2      ;DRAW
```

```
626C: A9 FF                      LDA   #$FF
626E: 20 A8 FC                   JSR   WAIT
6271: 20 20 63                   JSR   INITE4
6274: 20 A7 62                   JSR   DRAWE2      ;ERASE
6277: 60                         RTS
                 *************************
6278: AC 09 60   DRAWE1          LDY   HORIZB      ;ROUTINE FOR FIRST 3
627B: AE 36 60                   LDX   ELINE            EXPLOSION SHAPES
627E: BD 66 67                   LDA   HI,X
6281: 85 1B                      STA   HIGH
6283: BD 26 68                   LDA   LO,X
6286: 85 1A                      STA   LOW
6288: AE 0C 60                   LDX   XCOUNT
628B: B1 1A                      LDA   (LOW),Y
628D: 5D 36 65                   EOR   ESHAPE,X
6290: 91 1A                      STA   (LOW),Y
6292: EE 0C 60                   INC   XCOUNT
6295: EE 36 60                   INC   ELINE
6298: AD 36 60                   LDA   ELINE
629B: CD 38 60                   CMP   EDEPTH
629E: 90 D8                      BLT   DRAWE1
62A0: AD 37 60                   LDA   ELINEA
62A3: 8D 36 60                   STA   ELINE
62A6: 60                         RTS
                 ***********************
62A7: AC 09 60   DRAWE2          LDY   HORIZB      ;ROUTINE FOR FOURTH
62AA: AE 36 60                   LDX   ELINE            EXPLOSION SHAPE
62AD: BD 66 67                   LDA   HI,X
62B0: 85 1B                      STA   HIGH
62B2: BD 26 68                   LDA   LO,X
62B5: 85 1A                      STA   LOW
62B7: AE 0C 60                   LDX   XCOUNT
62BA: B1 1A                      LDA   (LOW),Y
62BC: 5D 36 65                   EOR   ESHAPE,X
62BF: 91 1A                      STA   (LOW),Y
62C1: EE 0C 60                   INC   XCOUNT
62C4: C8                         INY
62C5: AE 0C 60                   LDX   XCOUNT
62C8: B1 1A                      LDA   (LOW),Y
62CA: 5D 36 65                   EOR   ESHAPE,X
62CD: 91 1A                      STA   (LOW),Y
62CF: EE 0C 60                   INC   XCOUNT
62D2: EE 36 60                   INC   ELINE
62D5: AD 36 60                   LDA   ELINE
62D8: CD 38 60                   CMP   EDEPTH
62DB: 90 CA                      BLT   DRAWE2
62DD: AD 37 60                   LDA   ELINEA
62E0: 8D 36 60                   STA   ELINE
62E3: 60                         RTS
                 ************************
62E4: A9 00      INITE1          LDA   #$00        ;INITIALIZE FIRST EXPLOSION
62E6: 8D 0C 60                   STA   XCOUNT
62E9: A9 09                      LDA   #$09
62EB: 8D 37 60                   STA   ELINEA
62EE: 8D 36 60                   STA   ELINE
62F1: 18                         CLC
62F2: 69 05                      ADC   #$05
62F4: 8D 38 60                   STA   EDEPTH
62F7: 60                         RTS
```

```
62F8: A9 05     INITE2  LDA  #$05       ;INITIALIZE SECOND EXPLOSION
62FA: 8D 0C 60          STA  XCOUNT
62FD: A9 09             LDA  #$09
62FF: 8D 37 60          STA  ELINEA
6302: 8D 36 60          STA  ELINE
6305: 18                CLC
6306: 69 05             ADC  #$05
6308: 8D 38 60          STA  EDEPTH
630B: 60                RTS
630C: A9 0A     INITE3  LDA  #$0A       ;INITIALIZE THIRD EXPLOSION
630E: 8D 0C 60          STA  XCOUNT
6311: A9 05             LDA  #$05
6313: 8D 37 60          STA  ELINEA
6316: 8D 36 60          STA  ELINE
6319: 18                CLC
631A: 69 08             ADC  #$08
631C: 8D 38 60          STA  EDEPTH
631F: 60                RTS
6320: A9 12     INITE4  LDA  #$12       ;INITIALIZE FOURTH EXPLOSION
6322: 8D 0C 60          STA  XCOUNT
6325: A9 01             LDA  #$01
6327: 8D 37 60          STA  ELINEA
632A: 8D 36 60          STA  ELINE
632D: 18                CLC
632E: 69 0C             ADC  #$0C
6330: 8D 38 60          STA  EDEPTH
6333: 60                RTS
                ************************
6334: AD 39 60  SCORE   LDA  SUM        ;IF SUM = 0 AND
6337: C9 FF             CMP  #$FF         DECREMENTED TO #$FF
6339: D0 17             BNE  C1           THEN
633B: A9 09             LDA  #$09         SET SUM TO
633D: 8D 39 60          STA  SUM          #$09 AND
6340: CE 3A 60          DEC  COUNTER      DECREMENT COUNTER
6343: AD 3A 60          LDA  COUNTER      IF COUNTER = 0
6346: C9 FF             CMP  #$FF         AND DECREMENTED TO #$FF
6348: D0 08             BNE  C1           THEN
634A: A9 00             LDA  #$00         SET COUNTER
634C: 8D 3A 60          STA  COUNTER      AND SUM
634F: 8D 39 60          STA  SUM          EQUAL TO ZERO
6352: AD 39 60  C1      LDA  SUM        ;GET SUM
6355: C9 0A             CMP  #$0A       ;LESS THAN 10?
6357: 90 0C             BLT  C10        ;IF YES, BRANCH TO PRINT
6359: EE 3A 60          INC  COUNTER    ;IF NO, INCREMENT COUNTER AND
635C: AD 39 60          LDA  SUM         SUBTRACT 10 FROM SUM
635F: 38                SEC
6360: E9 0A             SBC  #$0A
6362: 8D 39 60          STA  SUM
6365: 0A        C10     ASL             ;MULTIPLY BY 8
6366: 0A                ASL
6367: 0A                ASL
6368: AA                TAX
6369: A0 13             LDY  #$13       ;POSITION FOR FIRST DIGIT
636B: 20 9D 63          JSR  PRINT      ;PRINT DIGIT
636E: AD 3A 60          LDA  COUNTER    ;GET COUNTER
6371: C9 0A             CMP  #$0A       ;LESS THAN 10?
6373: B0 0A             BGE  STOP1      ;IF NO, PRINT 100 AND STOP GAME
6375: 0A                ASL             ;IF YES, MULTIPLY BY 8
6376: 0A                ASL
```

```
6377: 0A                    ASL
6378: AA                    TAX
6379: A0 12                 LDY  #$12        ;POSITION OF SECOND DIGIT
637B: 20 9D 63              JSR  PRINT       ;PRINT DIGIT
637E: 60                    RTS
                   ************************
                   ** PRINTS 100 AND STOPS GAME **
637F: A2 08        STOP1    LDX  #$08        ;ACCESSES DIGIT "1"
6381: A0 11                 LDY  #$11        ;BYTE POSITION OF LEFTMOST DIGIT
6383: 20 9D 63              JSR  PRINT       ;PRINT DIGIT
6386: A2 00                 LDX  #$00        ;ACCESSES DIGIT "0"
6388: A0 12                 LDY  #$12        ;BYTE POSITION OF MIDDLE DIGIT
638A: 20 9D 63              JSR  PRINT       ;PRINT DIGIT
638D: A0 13                 LDY  #$13        ;BYTE POSITION OF FIRST DIGIT
638F: 20 9D 63              JSR  PRINT       ;PRINT DIGIT
6392: 2C 00 C0     STOP     BIT  $C000       ;ANY KEY PRESSED?
6395: 10 FB                 BPL  STOP        ;IF NO, BRANCH BACK & WAIT
6397: 2C 10 C0              BIT  $C010       ;IF YES, CLEAR KEYBOARD STROBE AND
639A: 4C 57 60              JMP  PGM            START PROGRAM OVER
                   **************************
639D: BD CE 63     PRINT    LDA  NSHAPE,X    ;RETRIEVE NUMBER SHAPE
63A0: 99 D0 23              STA  $23D0,Y     ;LINE #$B8 (184)
63A3: BD CF 63              LDA  NSHAPE+1,X
63A6: 99 D0 27              STA  $27D0,Y     ;LINE #$B9 (185)
63A9: BD D0 63              LDA  NSHAPE+2,X
63AC: 99 D0 2B              STA  $2BD0,Y     ;LINE #$BA (186)
63AF: BD D1 63              LDA  NSHAPE+3,X
63B2: 99 D0 2F              STA  $2FD0,Y     ;LINE #$BB (187)
63B5: BD D2 63              LDA  NSHAPE+4,X
63B8: 99 D0 33              STA  $33D0,Y     ;LINE #$BC (188)
63BB: BD D3 63              LDA  NSHAPE+5,X
63BE: 99 D0 37              STA  $37D0,Y     ;LINE #$BD (189)
63C1: BD D4 63              LDA  NSHAPE+6,X
63C4: 99 D0 3B              STA  $3BD0,Y     ;LINE #$BE (190)
63C7: BD D5 63              LDA  NSHAPE+7,X
63CA: 99 D0 3F              STA  $3FD0,Y     ;LINE #$BF (191)
63CD: 60                    RTS
                   ****************************
63CE: 00 1C 22     NSHAPE   HEX  001C22222222221C     ;NUMBER SHAPES - "0"
63D1: 22 22 22 22 1C
63D6: 00 08 0C              HEX  00080C080808081C     ;"1"
63D9: 08 08 08 08 1C
63DE: 00 1C 22              HEX  001C22201008043E     ;"2"
63E1: 20 10 08 04 3E
63E6: 00 1C 22              HEX  001C22201C20221C     ;"3"
63E9: 20 1C 20 22 1C
63EE: 00 10 18              HEX  00101814123E1010     ;"4"
63F1: 14 12 3E 10 10
63F6: 00 3E 02              HEX  003E021E2020201E     ;"5"
63F9: 1E 20 20 20 1E
63FE: 00 1C 22              HEX  001C22021E22221C     ;"6"
6401: 02 1E 22 22 1C
6406: 00 3E 20              HEX  003E201008040404     ;"7"
6409: 10 08 04 04 04
640E: 00 1C 22              HEX  001C22221C22221C     ;"8"
6411: 22 1C 22 22 1C
6416: 00 1C 22              HEX  001C22223C20221C     ;"9"
6419: 22 3C 20 22 1C
641E: 00 0E 01     MSHAPE1  HEX  000E01000E01000E01   ;MAN SHAPE TABLES
```

```
6421: 00 0E 01 00 0E 01
6427: 00 44 01  460          HEX  004401007F00601F00
642A: 00 7F 00 60 1F 00
6430: 30 1F 00  461          HEX  301F00181F00001F00
6433: 18 1F 00 00 1F 00
6439: 00 1F 00  462          HEX  001F00001B00403100
643C: 00 1B 00 40 31 00
6442: 60 60 00  463          HEX  606000
6445: 00 1C 02  464 MSHAPE2  HEX  001C02001C02001C02
6448: 00 1C 02 00 1C 02
644E: 00 08 03  465          HEX  000803007E01003E00
6451: 00 7E 01 00 3E 00
6457: 00 3F 00  466          HEX  003F00403F00003E00
645A: 40 3F 00 00 3E 00
6460: 00 3E 00  467          HEX  003E00003600003600
6463: 00 36 00 00 36 00
6469: 00 63 00  468          HEX  006300
646C: 00 38 04  469 MSHAPE3  HEX  003804003804003804
646F: 00 38 04 00 38 04
6475: 00 10 06  470          HEX  001006007C03007C00
6478: 00 7C 03 00 7C 00
647E: 00 7C 00  471          HEX  007C00007E00007C00
6481: 00 7E 00 00 7C 00
6487: 00 38 00  472          HEX  003800003800006C00
648A: 00 38 00 00 6C 00
6490: 00 46 01  473          HEX  004601
6493: 00 70 08  474 MSHAPE4  HEX  007008007008007008
6496: 00 70 08 00 70 08
649C: 00 20 0C  475          HEX  00200C007807007801
649F: 00 78 07 00 78 01
64A5: 00 78 01  476          HEX  007801007801007801
64A8: 00 78 01 00 78 01
64AE: 00 70 00  477          HEX  007000007000007000
64B1: 00 70 00 00 70 00
64B7: 00 70 00  478          HEX  007000
64BA: 00 60 11  479 MSHAPE5  HEX  006011006011006011
64BD: 00 60 11 00 60 11
64C3: 00 40 18  480          HEX  00401800700F007003
64C6: 00 70 0F 00 70 03
64CC: 00 70 03  481          HEX  007003007803007003
64CF: 00 78 03 00 70 03
64D5: 00 60 01  482          HEX  006001006001003003
64D8: 00 60 01 00 30 03
64DE: 00 18 06  483          HEX  001806
64E1: 00 40 23  484 MSHAPE6  HEX  004023004023004023
64E4: 00 40 23 00 40 23
64EA: 00 00 31  485          HEX  00003100601F006007
64ED: 00 60 1F 00 60 07
64F3: 00 70 07  486          HEX  007007007807006007
64F6: 00 78 07 00 60 07
64FC: 00 60 07  487          HEX  006007006006006006
64FF: 00 60 06 00 60 06
6505: 00 30 0C  488          HEX  00300C
6508: 00 00 47  489 MSHAPE7  HEX  000047000047000047
650B: 00 00 47 00 00 47
6511: 00 00 62  490          HEX  00006200403F00700F
6514: 00 40 3F 00 70 0F
651A: 00 58 0F  491          HEX  00580F004C0F00400F
```

```
651D: 00 4C 0F 00 40 0F
6523: 00 40 0F  492          HEX  00400F00400D006018
6526: 00 40 0D 00 60 18
652C: 00 30 30  493          HEX  003030
652F: 01        494 BSHAPE1  HEX  01               ;BULLET SHAPES
6530: 02        495 BSHAPE2  HEX  02
6531: 04        496 BSHAPE3  HEX  04
6532: 08        497 BSHAPE4  HEX  08
6533: 10        498 BSHAPE5  HEX  10
6534: 20        499 BSHAPE6  HEX  20
6535: 40        500 BSHAPE7  HEX  40
6536: 28 22 1A  501 ESHAPE   HEX  28221A2514       ;EXPLOSION SHAPES  - NO. 1
6539: 25 14
653B: 2C 52 44  502          HEX  2C5244320C          ;NO. 2
653E: 32 0C
6540: 38 3E 7F  503          HEX  383E7F7E7E3F3F1C    ;NO. 3
6543: 7E 7E 3F 3F 1C
6548: 18 06 7C  504          HEX  18067C0F7C3F7E3F    ;NO. 4
654B: 0F 7C 3F 7E 3F
6550: 7C 7F 7C  505          HEX  7C7F7C3F7E3F7F1F
6553: 3F 7E 3F 7F 1F
6558: 7E 0F 7C  506          HEX  7E0F7C1F700F4003
655B: 1F 70 0F 40 03
                    BYTETBL
                    OFFSET
                    HI
                    LO


2278 bytes


Symbol table - numerical order:

   LOW      =$1A      HIGH     =$1B      MLINE    =$6003    MLINEA   =$6004
   BLINE    =$6005    DEPTH    =$6006    MHORIZ   =$6007    BHORIZ   =$6008
   HORIZB   =$6009    HORIZM   =$600A    BULON    =$600B    XCOUNT   =$600C
   DELAY    =$600D    BTEMP    =$600E    MTEMP    =$600F    ELINE    =$6036
   ELINEA   =$6037    EDEPTH   =$6038    SUM      =$6039    COUNTER  =$603A
   MSHPADR  =$603B    BSHPADR  =$6049    PGM      =$6057    CLR1     =$606B
   CLR      =$606F    LN       =$6091    LN1      =$60A8    PADDLE   =$60B8
   BULLET1  =$60D6    BULLET   =$60E1    TOP      =$6103    MINITIAL=$6109
   BINITIAL=$6118     SINITIAL=$6123     PR       =$612E    PDLE     =$6137
   PDLE1    =$614D    LOAD     =$6167    MDRAW    =$6172    MDRAW1   =$6177
   LOADBUL  =$61BC    BDRAW    =$61DF    NOHIT    =$61FB    COLLISION=$6203
   BXDRAW   =$621B    EXPLODE  =$6233    DRAWE1   =$6278    DRAWE2   =$62A7
   INITE1   =$62E4    INITE2   =$62F8    INITE3   =$630C    INITE4   =$6320
   SCORE    =$6334    C1       =$6352    C10      =$6365    STOP1    =$637F
   STOP     =$6392    PRINT    =$639D    NSHAPE   =$63CE    MSHAPE1  =$641E
   MSHAPE2  =$6445    MSHAPE3  =$646C    MSHAPE4  =$6493    MSHAPE5  =$64BA
   MSHAPE6  =$64E1    MSHAPE7  =$6508    BSHAPE1  =$652F    BSHAPE2  =$6530
   BSHAPE3  =$6531    BSHAPE4  =$6532    BSHAPE5  =$6533    BSHAPE6  =$6534
   BSHAPE7  =$6535    ESHAPE   =$6536    BYTETBL  =$6560    OFFSET   =$6663
   HI       =$6766    LO       =$6826    GRAPHICS=$C050     MIXOFF   =$C052
   PAGE1    =$C054    HIRES    =$C057    BUTTON   =$C061    PREAD    =$FB1E
   WAIT     =$FCA8
```

The protocols presented in this chapter are not the end-all of scoring routines. I know of at least two others that more or less accomplish the same purpose and I'm sure there are still others lurking in programs somewhere. Perhaps you could devise a better routine yourself. Why not give it a try? If you come up with something better, fame, fortune, and members of the opposite sex (or the same sex?) await you.

9

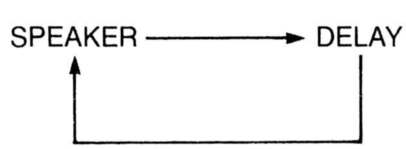
Sound Generation: Explosions and Clickety-Clicks

Clickety-click, buzz and wham
Puckety-puckety, pft and slam
Pow and bang
Whoosh and clang
Tinkely-tink, whir and blam.

Sound generation routines are among the easiest to explain but the hardest to apply, at least in game programs, requiring a great deal of trial and error and just plain all around fiddling. This is why other books on assembly language hi-res graphics and most commercial hi-res graphics utility programs omit the subject entirely. Well, there's no getting around it so let's jump in. We'll discuss first the principles of sound generation on the Apple II and then see how to apply these principles to our game program.

THE APPLE SPEAKER AND SOUND GENERATION

Somewhere in your little tan Apple box is what is laughingly called a loudspeaker. Its size is so small it gives new meaning to the term "low fidelity." However, it is capable of producing sounds, if not music. It does this in the following way. The speaker cone is in one of two positions, in or out. By accessing a soft switch located at $C030, the cone changes position thereby pushing air and producing a sound wave. When the speaker is accessed just once, a click can be heard if you listen carefully. By accessing the speaker in rapid succession, tones are produced; the more frequent the access, the higher the tone or pitch. The basic tone-producing cycle is as follows:



By altering the delay time, different pitches are produced, ranging anywhere from low-pitched clicks (long delay) to high-pitched tones (short delay).

Writing an assembly language program to produce the cycle depicted above is easy to do, especially for us experts, but something else is required lest the tone continue indefinitely (you could always stop the program or pull the plug and lucky Apple IIc owners could always turn down the volume, but let's do it properly). The point is that the cycle has to be interrupted so that we can specify the tone's duration. Let's look at the following program to see how it's done.

```
]PROGRAM 9-1
:ASM
                 1     **** SOUND ****
                 2              ORG  $6000
6000: 4C 04 60   3              JMP  PGM
                 4     DELAY    DS   1
                 5     SPEAKER  =    $C030
                 6     WAIT     =    $FCA8
6004: A9 60      7     PGM      LDA  #$60
6006: 8D 03 60   8              STA  DELAY
6009: A0 02      9              LDY  #$02
600B: 2C 30 C0  10     SOUND    BIT  SPEAKER
600E: AD 03 60  11              LDA  DELAY
6011: 20 A8 FC  12              JSR  WAIT
6014: 88        13              DEY
6015: D0 F4     14              BNE  SOUND
6017: 4C 04 60  15              JMP  PGM


--End assembly--

26 bytes


Symbol table - numerical order:

   DELAY   =$6003      PGM     =$6004      SOUND   =$600B      SPEAKER =$C030
   WAIT    =$FCA8
```


The program loads DELAY with #$60 and Y with #$02. At SOUND, the speaker is accessed, using BIT instead of LDA just because I feel like it ($C030 must be accessed either with LDA or BIT, not STA). There is then a time delay followed by a DEY and BNE SOUND. A BNE that doesn't follow a comparison instruction simply means branch if the previous operation results in a non-zero; in this case, branch if Y hasn't reached zero yet. Because Y is now 1, the program branches for another speaker access and delay. DEY now decrements Y to zero so the branch is not taken and the program jumps to START for another round. What's happening here is that the value in DELAY is specifying the pitch, i.e., the time between speaker accesses, while Y specifies the duration. The duration effect is not readily apparent in this program, because at the end of each tone pulse, the program branches immediately back to the beginning for another cycle. If we replace line 15 with RTS, we would then produce a single tone whose duration would be more obviously controlled by Y.

One problem with this type of routine should be mentioned, although it has no bearing on our game program. The duration of the tone depends not only on Y but also on DELAY, because DELAY contributes to the overall execution time for the routine and, thus, a given Y cannot be used to produce different pitched tones of equal duration. (By the way, have you ever noticed that tone is an anagram of note? I only mention this to take your mind off the duration control problem.)

To see the effect of the time delay on pitch, run Program 9-1 with different values in DELAY. A value of #$60 produces a low, rapid clicking. If we increase the delay, the clicks become lower and slower—#$90 produces a kind of put-put and #$BB a sort of hoppity-hop (this is by far the hardest part of writing this book, trying to find adjectives to describe these sounds!). Decreasing the delay produces higher pitches and more rapid clicking. A value of #$40 produces a buzz, and it's only when we get to #$30 or below that we hear something that resembles a musical tone. A value of #$05 results in a very high pitched tone, just barely audible and just barely bearable. Try #$01 and drive every dog in the neighborhood crazy. Isn't this fun? I hope you're enjoying it, because now we're getting to the sticky part.

INTEGRATING SOUND EFFECTS INTO THE GAME PROGRAM

Although we won't be discussing the game program as such until the next chapter, we know enough about the game already to allow us to apply the principles of sound generation to the development of sound effects. The game itself is relatively simple and this limits our options. The sound effects I've decided to include are an explosion sound when a plane is hit and some sort of sound when a plane is traversing the screen.

One problem with integrating sound effects into any type of program is that any sort of sound, except individual clicks, requires a time delay between speaker accesses and we have to be careful that our sound routine doesn't result in an unwanted program interruption. The solution to this problem is to insert sound routines where program delays already exist.

Let's discuss the explosion sound first. In the game program, as in Program 8-1, a collision sends the program to the EXPLODE subroutine where the explosion shapes are drawn and erased. Remember that between each draw and erase, we inserted a time delay. What I've done is substitute the explosion sound routine for the first delay, i.e., between drawing and erasing the first explosion shape. In other words, the sound routine itself provides the delay—in this way we've added an extra feature to the program without altering its execution time. The sound routine itself is listed below:

```
SOUND   LDY #$02
SOUND1  BIT SPEAKER
        LDA #$60
        JSR WAIT
        DEY
        BNE SOUND1
        RTS
```

This routine produces a single tone with a delay of #$60 between speaker accesses. The total delay for this routine can be calculated as follows. The LDA #$60, JSR WAIT is accessed twice (Y = 2). Two times #$60 equals 2 times 96 = 192 or #$C0. The original delay time was #$BB or 187. Thus, even adding a little extra time for the execution of the sound routine, we see that the total delay is very close to what we had originally.

It might seem, on paper at least, that a single tone is hardly appropriate for an explosion sound but if you run the game program you'll see that it works. That's why so much fiddling is required—what looks bad on paper may be perfectly alright in a program and, unfortunately, vice versa.

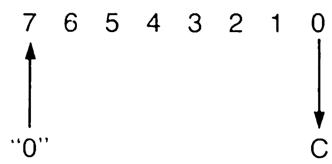
In spite of the fact that the routine works, I'm sure that with a little extra fiddling, you or I could come up with something better. Instead of me doing it for you, here is an opportunity for you to display your expertise and imagination (if you think I'm trying to wriggle out of this, you're right). How can we do this? Well, we could try to alter the tone by raising the pitch and duration. Doing this would not increase the total delay time because raising the pitch means less delay between speaker accesses. We could try inserting other tones in other program locations—obvious places would be the time delays between drawing and erasing the other explosion shapes. We could try—well, as I told you in the beginning, there's a lot of trial and error involved in this process, and so on some cold February night, with a blizzard raging, nothing on television, the kids asleep, the dog at the vet, and your wife/hubby in Hoboken for the annual meeting of the International Computer Widows/Widowers Association, give it a try. You have nothing to lose except your sanity.

Now we get to the plane sound. The plane as drawn looks like it's jet- or rocket-powered and so one might imagine that we should strive for something like a whooshing sound. However, computer game programmers are allowed the equivalent of poetic license, which means if it's too hard, we'll do something else. The plane is, in fact, powered by an electric motor. Why? Because I say so—after all, I am its creator (there's nothing like learning assembly language to give one a feeling of omnipotency)—and besides, the motor sound effect somehow seems to work. The sound we want then is a kind of clickety-click (there are those adjectives again) and one way to accomplish this is by clicking the speaker just once each time a plane is moved one bit position. The delay between clicks is accomplished by the program itself as it loops from one plane move to the next. However, when I tried this in the game program I wasn't entirely satisfied because the clicks were too rapid, and so I decided instead to have the speaker click every other plane move. The technique for doing this illustrates a method generally applicable to any situation where we want to access a routine every other cycle, so let's discuss the details.

The Apple II microprocessor contains another register besides the Accumulator, X, and Y, called the Status Register, which can also hold just a single byte. In contrast to the other registers, however, the Status Register is not used to store numbers per se, but rather to indicate certain conditions by having each bit contain a 1 or 0. I'm not going to discuss all the functions of the Status Register bits but the one bit I do want to discuss is called the Carry bit. One function of the Carry bit is to indicate an overflow when a number is added to #$FF in the Accumulator. For example, adding #$01 to #$FF zeros the Accumulator and sets the Carry bit to 1. This is why the instruction ADC (ADd with Carry) requires a prior CLC (CLear Carry) and why SBC (SuBtract with Carry) requires a prior SEC (SEt Carry). Another function of the Carry bit is as an indicator in comparisons. For example, CMP compares a value to the value in the Accumulator. If the value in the Accumulator is less than the compared value, the Carry bit is cleared (0); if more, the Carry bit is set (1). This is why the pseudo-op BLT (Branch if Less Than) is used in some Assemblers in place of BCC (Branch on Carry Clear), and BGE (Branch if Greater or Equal) in place of BCS (Branch on Carry Set).

Now we're ready to see how we can use the Carry bit for our every-other-cycle click routine.

The instruction LSR (Logical Shift Right) moves each bit in the Accumulator one position to the right—a zero is moved into bit 7 and bit 0 moves into the Carry. Note that some assemblers require A in the operand column, i.e., LSR A.



LSR can be used to test if the number in the Accumulator is odd or even. If even, bit 0 (this is the 1's column) must contain 0 and after LSR, the Carry bit will be clear; if odd, bit 0 must contain 1 and after LSR the Carry bit will be set. Thus, a BCC will branch the program if the number is even and not branch if the number is odd (here we're using BCC instead of BLT because the standard mnemonic reminds us what the instruction is doing).

Now let's see how to use LSR to alternate the speaker clicks. First, somewhere in the beginning of the program we define a memory location DE. Then, in the MAIN PROGRAM at the point where we draw a plane with JSR PDRAW, we include the following routine:

```
        JSR PDRAW      ;DRAW PLANE
        INC DE
        LDA DE
        ***            ;CARRY = 0 IF DE IS EVEN
        LSR            ;CARRY = 1 IF DE IS ODD
        BCC BUL        ;SKIP NEXT LINE IF CARRY = 0
        BIT SPEAKER    ;CLICK SPEAKER
BUL     etc.
```

Every time a plane is drawn, DE changes from odd to even or vice versa and thus the speaker is accessed only every other plane draw. Because this routine clicks the speaker just once, no time delay is involved (except for the time it takes to run the routine) and the program execution time is not noticeably affected. DE does not have to be set to any particular number in the beginning of the program, as the actual value in DE is immaterial for the odd-even cycle. Note also that DE never fills up because when it reaches #$FF it simply wraps around to #$00. In addition, notice that we first load DE into the Accumulator and then do an LSR on the Accumulator contents. The LSR instruction can have a memory location as the operand, but if we perform an LSR DE directly without loading DE into the Accumulator, DE would itself be changed and this would interfere with the odd-even cycling.

This brings us to the end of our preparatory chapters. In the next chapter we will see how to assemble the final game program.

10

Putting It All Together: The Game

There once was a girl from Sydney
Who could . . . (never mind).

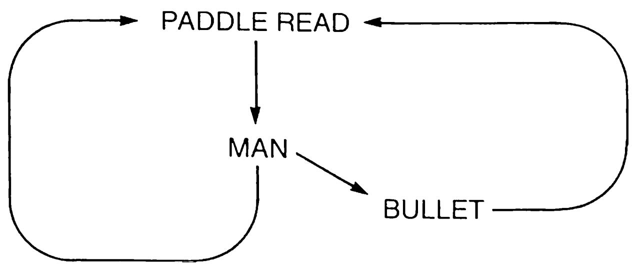
Our goal is finally in sight. All we need do now is to take our expert knowledge of assembly language programming and the routines we've already developed and assemble them into the final game program, but this is easier said than done as we'll soon see.

Essentially what we're going to do is merge Programs 8-1 and 5-1, and add sound routines and a few other embellishments. Before we do this, however, a brief description of the game is in order. A man will move along a bottom line, his movement controlled by a paddle or joystick. Planes will appear, with appropriate sound effects, one at a time near the top of the screen, moving left to right all at the same screen line position. A bullet can be fired by pressing the paddle or joystick button. If a plane isn't hit by a bullet, it continues to the end of the screen where it is erased and a new plane then reappears for another screen traversal. If a plane is hit, it explodes with a (sort of) bang, the score is incremented by 1 and another plane then appears at the left border. The game stops when the score reaches 100 or when 100 planes have appeared. Thus, if 10 planes are missed, the game will stop at a score of 90. In this way, a player can tell how close he came to the ideal of hitting all 100 planes. The game can be restarted by pressing any key.

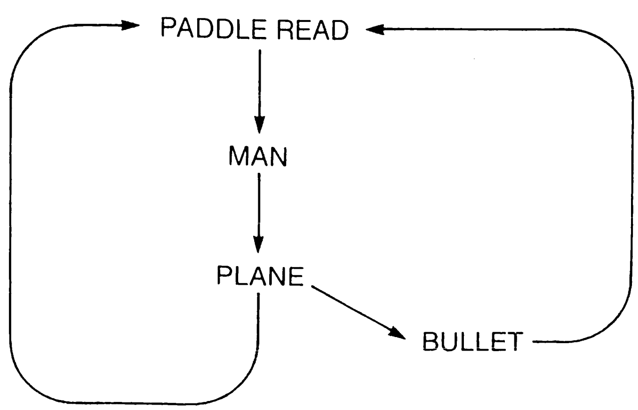
Now to the heart of the matter. As mentioned in a previous chapter, when dealing with a relatively complicated program, it is essential to design the flowchart first, leaving the details to later. The fact that we already have most of the details is of no matter. It is merely a consequence of the fact that this book is a teaching exercise—an experienced programmer would start with this chapter first and work backwards, so to speak, to fill in the details.

The main problem in designing a game program is ordering the routines in such a way that the desired simulation is achieved. Remember that a computer can do only one thing at a time and in the final program we might want to simulate, at some points at least, simultaneous events, and some events must follow or precede others. We have to consider which shapes to draw first, which to draw last, when to erase, where to insert the paddle read, score, and explosions, etc. The ordering of routines then is the salient dictum.

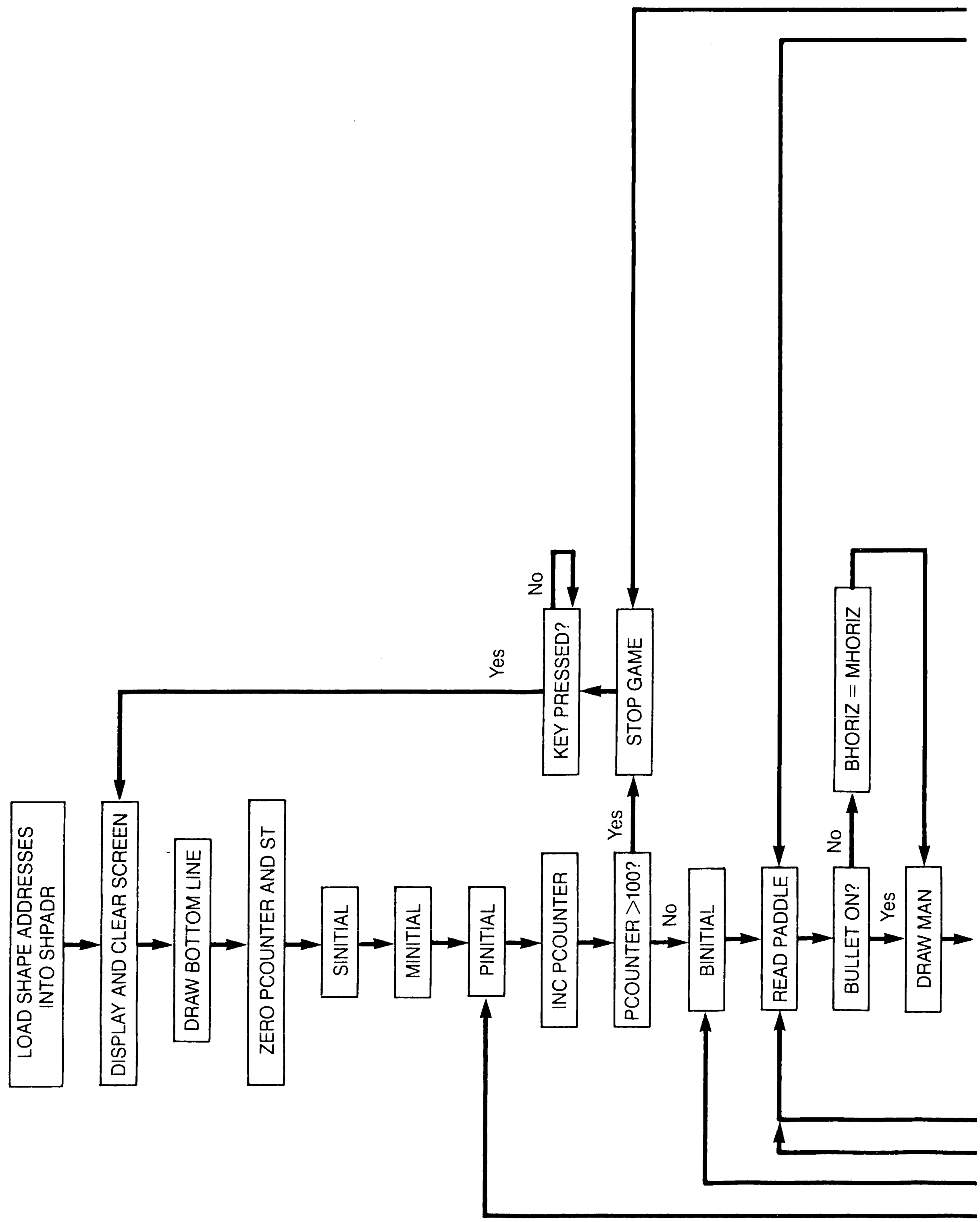
Remember that in Chapter 6 we discussed for the first time how to design a program with two shapes moving at the same time, the man and bullet. The overall design can be depicted as follows:

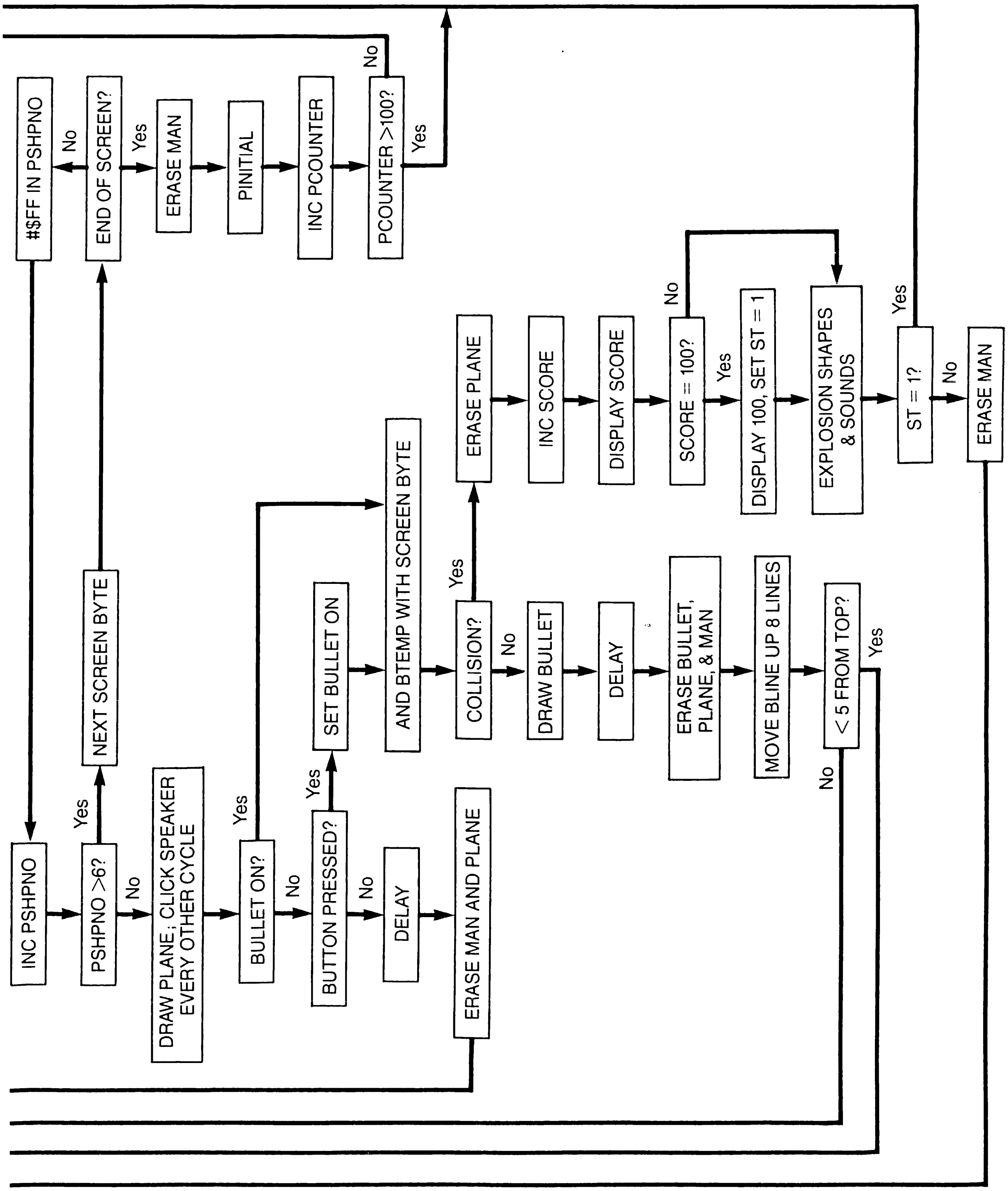


The program continues in this loop indefinitely even if a bullet isn't drawn or if the paddle doesn't change position. The important point is that we created the illusion that the bullet moves at the same time the man's movement is controlled by the paddle even though, of course, each man and bullet move is a separate event. This is a direct consequence of the speed of the program—the time between the bullet move and the paddle-controlled move is so small as to produce the illusion of simultaneity. The game program uses the same principle except here we're drawing a plane after the man draw:



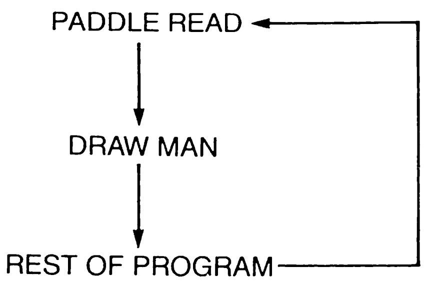
Again, the speed of the program allows us to create the illusion of three shapes moving at the same time. Now, with all this in mind, let's examine the flowchart for the game program.


LOAD SHAPE ADDRESSES INTO SHPADR
DISPLAY AND CLEAR SCREEN
DRAW BOTTOM LINE
ZERO PCOUNTER AND ST
SINITIAL
MINITIAL
PINITIAL
INC PCOUNTER
PCOUNTER >100?
Yes
STOP GAME
KEY PRESSED?
Yes
No
No
BINITIAL
READ PADDLE
BULLET ON?
No
BHORIZ = MHORIZ
Yes
DRAW MAN


INC PSHPNO
#$FF IN PSHPNO
PSHPNO >6?
Yes
No
NEXT SCREEN BYTE
END OF SCREEN?
ERASE MAN
PINITIAL
INC PCOUNTER
PCOUNTER >100?
DRAW PLANE; CLICK SPEAKER EVERY OTHER CYCLE
BULLET ON?
BUTTON PRESSED?
SET BULLET ON
DELAY
AND BTEMP WITH SCREEN BYTE
ERASE MAN AND PLANE
COLLISION?
ERASE PLANE
DRAW BULLET
INC SCORE
DELAY
DISPLAY SCORE
ERASE BULLET, PLANE, & MAN
SCORE = 100?
MOVE BLINE UP 8 LINES
DISPLAY 100, SET ST = 1
< 5 FROM TOP?
EXPLOSION SHAPES & SOUNDS
ST = 1?
ERASE MAN

In the program's beginning, we take care of the "housekeeping" chores—the shape addresses are assembled into shape address tables, the screen is cleared and displayed, the bottom line drawn, counters are zeroed, and we initialize the score, man, plane, and bullet. Let's stop here to discuss a point we haven't seen before. Remember that we want to keep track of how many planes are drawn so we can stop the program when 100 planes have appeared (if the perfect score of 100 has not been attained). We do this in the PINITIAL routine because this routine is accessed when, and only when, a new plane is drawn, either after a plane has been hit or when a plane has reached the end of the screen (and of course for the first plane draw). We accomplish this by incrementing PCOUNTER (initially set to zero at the beginning) for each access to PINITIAL and asking whether PCOUNTER contains a value less than 101. If it does, we continue—if it doesn't, we stop the program (we don't do a comparison to 100 because we want the 100th plane to be drawn).

The program then continues with a paddle read and man draw. If there is such a thing as a grand design for a program, we can illustrate it in the following diagram:



Everything else in the program we want to do, drawing the planes and bullets, keeping score, displaying explosions, all the sound effects, reinitializing, are all done in between paddle reads. The program doesn't have to be designed this way—it's just that the program is fast enough so that paddle reads do not have to be done more frequently. The illusion we're striving for (and attaining) is that the man's position is always responsive to the paddle regardless of whatever else is going on. (But note the caveat discussed in Chapter 7—the man's position becomes momentarily unresponsive to the paddle during the explosion shapes display; but note, too, as also discussed in Chapter 7, that this delay is hardly noticeable and could be eliminated by inserting other paddle reads between drawing and erasing the explosion shapes.)

The paddle read routine, you will remember, also contains a "bullet on?" question. If the bullet is not on, BHORIZ is set equal to MHORIZ so that when the bullet is fired, it will be aligned with the man. If the bullet is already on, this step is skipped to allow the bullet to move up independent of the man's position.

After the paddle read, the man is drawn and then we prepare to draw the plane. Here we access the plane shape number a little differently than before for programming convenience. We first increment PSHPNO. Because we want the first shape, PSHPNO should contain #$00 after we increment. This is why PSHPNO is loaded with #$FF in PINITIAL—incrementing #$FF wraps the value around to #$00. We then ask if the value in PSHPNO is greater than 6; i.e., have we finished all seven shapes? If no, we continue and draw the plane, clicking the speaker every other cycle. If yes, we move to the next screen byte and ask if we've reached the end of the screen. If no, we load PSHPNO with #$FF again and loop back to INC PSHPNO in preparation for the next plane draw, this time with shape 0 at the next screen byte position. If we have reached the end of the screen, we erase the man (the plane has already been erased by the DRAW-ERASE protocol), and initialize the plane again, thereby also incrementing PCOUNTER. We then test PCOUNTER to see if 100 planes have appeared—if yes, we stop the program; if no, we continue with another paddle read which draws the man, and then the plane is drawn again.

After the plane is drawn, we ask if the bullet is on or if the button is pressed. If neither, we skip the bullet draw routine entirely and erase the man and the plane in preparation for another paddle read—the man's position is updated and the plane moves over one bit. Note a general feature of the program; just before a paddle access, we always erase whatever shape (man, plane, or bullet) that happens to be on the screen, in preparation for the next move.

Supposing the button is pressed or the bullet is already on the screen—we then go to the bullet draw routine before we reaccess the paddle (if the button is pressed, the bullet on marker, BULON, is set). The BDRAW routine first does a collision test before the bullet is drawn. If there is no collision, the bullet is drawn and then the bullet, plane, and man are erased in preparation for the next paddle read. Before we access the paddle however, we move the next bullet position up eight lines and ask if it has reached the top (actually, within five lines of the top). If it hasn't, that's fine—we simply go back for another paddle read. If it has, we have to initialize the bullet first before the paddle read.

Suppose a collision is detected. Here I've changed the protocol slightly to produce what I think is a better display. You remember in Program 8-1 we displayed the explosion first and then the score. This was done so that when the score reached 100 (if all 100 planes were hit) the explosion would finish before the score indicated the program should stop; otherwise, we would be left with an unexploded plane on the screen. I found this delay in displaying the score unnerving—I want to see the score right after that plane is blasted! So here we're going to display the score first, before the explosion.

In the COLLISION routine, we first erase the plane (we always erase the shape that's hit), INC SUM, and then go to SCORE routine. In SCORE, if the count is 100, we print 100 in the display and then, instead of stopping the program, we load an indicator labeled ST with #$01. If the score is not 100, ST contains #$00. We then jump back to the COLLISION routine, draw and erase the explosion shapes with the accompanying sound effect, and test ST. If ST contains #$01, it means the score has reached 100 and we stop the program—remember we've already displayed and erased the explosion so we're not left with an unexploded plane. If ST contains #$00, the score has not reached 100 and we continue by erasing the man (there's no need to erase the bullet, as the collision test is done before the bullet is drawn), initializing the plane and bullet, and reading the paddle again.

We've now come to the end of the flowchart. A good way to check out a program, to make sure it's doing what we want it to do, is to run through the flowchart considering all possible routes, so let's do that now.

Situation—The man and plane are drawn and the bullet is not fired and is not on the screen.

Flow—After a delay, the man and plane are erased and the paddle read again for another man and plane draw. The plane moves across the screen and the man's movement is controlled by the paddle; nothing else happens.

Situation—The man and plane are drawn but PSHPNO indicates the next plane draw will reach the end of the screen.

Flow—The man is erased and the plane initialized so that the next plane will appear at the starting left border position. If 100 planes have appeared, the program stops; if not, the paddle is read again to update the man's position and the next man and plane are drawn.

Situation—The man and plane are drawn and the button is pressed.

Flow—BULON is set to indicate bullet on and the bullet draw routine is accessed. Because the bullet was not on when the paddle was read, BHORIZ is equal to MHORIZ and the bullet is fired from the man's position.

Situation—The man and plane are drawn and the bullet is on the screen.

Flow—BDRAW is accessed but BHORIZ is now independent of MHORIZ, so the bullet can move up independently of the man's position.

Situation—The man and plane are drawn, a bullet is on the screen but not yet at the top, and no collision is detected.

Flow—The bullet is drawn and after a delay, the bullet, man, and plane are erased. The paddle is read again to update the man shape position, the plane moves one bit position and the bullet moves up eight lines. This continues until the bullet has reached the top of the screen or until a plane is hit.

Situation—Same as above but the bullet has reached the top.

Flow—The bullet is initialized, which sets BULON to indicate the bullet is not on and the bullet will not be drawn unless the button is pressed. Thus, we're back to the situation where the man and plane are drawn but the bullet is not on.

Situation—The man and plane are drawn, a bullet is on the screen, and a collision is detected.

Flow—The plane is erased, SUM incremented, the score displayed, and the explosion shapes drawn and erased with the explosion sound effect. If the score is at 100, the program stops. If less than 100, the man is erased, the plane and bullet initialized, and the paddle read in preparation for another cycle.

Situation—A plane has been hit or has reached the end of the screen.

Flow—PCOUNTER is incremented for each such occurrence. After 100 such events, the program stops (unless stopped by the score reaching 100).

This takes us through essentially all the game assembly, as we already know most of the program details. In fact, there is only one minor detail that need be considered. Conditional branch instructions branch to program locations by relative rather than absolute addressing; i.e., the location to be branched to is not specified by a particular address but rather by the distance (in bytes) from the branch instruction. The branching distance is limited, however, by a maximum of 127 bytes forward or 128 bytes back. What do you do if you want to branch to a location outside these limits? Lines 172-174 and 365-368 in Program 10-1 illustrate the solution. In line 172, for example, what we would like to do is branch to BI with a BLT BI but BI is too far from the branch instruction. So, what we do instead is insert a short branch to a JMP instruction (JMP branches to absolute addresses and thus does not have any distance limitation). The routine is:

```
        BLT LONG
        ....
        ....
LONG    JMP BI
```

By the way, your assembler will tell you, with an error message, when you attempt to branch beyond the distance limits.

And now—fanfare please—it is with great pride (or at least some pride) and little trepidation that I hereby present THE GAME! (Whoops—it has no name! To enter the Name the Game contest, send $10 in cash, and also an entry if you like, to me, care of the publisher. The winner will receive a thank you note suitable for framing.)

```
]PROGRAM 10-1
:ASM
                     **** THE GAME! ****
                              ORG   $6000
6000: 4C 7C 60                JMP   PGM
                     MLINE    DS    1
                     MLINEA   DS    1
                     BLINE    DS    1
                     DEPTH    DS    1
                     MHORIZ   DS    1
                     BHORIZ   DS    1
                     HORIZB   DS    1
                     HORIZM   DS    1
                     BULON    DS    1
                     XCOUNT   DS    1
                     DELAY    DS    1
                     BTEMP    DS    1
                     MTEMP    DS    39
                     ELINE    DS    1
                     ELINEA   DS    1
                     EDEPTH   DS    1
                     SUM      DS    1
                     COUNTER  DS    1
                     DE       DS    1
                     PCOUNTER DS    1
                     PLINE    DS    1
                     PLINEA   DS    1
                     PBYTE    DS    1
                     PDEPTH   DS    1
                     PSHPNO   DS    1
                     PTEMP    DS    15
                     ST       DS    1
                     GRAPHICS =     $C050
                     MIXOFF   =     $C052
                     HIRES    =     $C057
                     PAGE1    =     $C054
                     HIGH     =     $1B
                     LOW      =     $1A
                     WAIT     =     $FCA8
                     PREAD    =     $FB1E
```

```
                  BUTTON   =    $C061      ;BUTTON 0
                  SPEAKER  =    $C030
                  *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                  *CONTINUE FOR ALL 7 SHAPES
6052: EE          MSHPADR  DFB  #<MSHAPE1
6053: 64                   DFB  #>MSHAPE1
6054: 15                   DFB  #<MSHAPE2
6055: 65                   DFB  #>MSHAPE2
6056: 3C                   DFB  #<MSHAPE3
6057: 65                   DFB  #>MSHAPE3
6058: 63                   DFB  #<MSHAPE4
6059: 65                   DFB  #>MSHAPE4
605A: 8A                   DFB  #<MSHAPE5
605B: 65                   DFB  #>MSHAPE5
605C: B1                   DFB  #<MSHAPE6
605D: 65                   DFB  #>MSHAPE6
605E: D8                   DFB  #<MSHAPE7
605F: 65                   DFB  #>MSHAPE7
6060: FF          BSHPADR  DFB  #<BSHAPE1
6061: 65                   DFB  #>BSHAPE1
6062: 00                   DFB  #<BSHAPE2
6063: 66                   DFB  #>BSHAPE2
6064: 01                   DFB  #<BSHAPE3
6065: 66                   DFB  #>BSHAPE3
6066: 02                   DFB  #<BSHAPE4
6067: 66                   DFB  #>BSHAPE4
6068: 03                   DFB  #<BSHAPE5
6069: 66                   DFB  #>BSHAPE5
606A: 04                   DFB  #<BSHAPE6
606B: 66                   DFB  #>BSHAPE6
606C: 05                   DFB  #<BSHAPE7
606D: 66                   DFB  #>BSHAPE7
606E: 30          PSHPADR  DFB  #<PSHAPE1
606F: 66                   DFB  #>PSHAPE1
6070: 3F                   DFB  #<PSHAPE2
6071: 66                   DFB  #>PSHAPE2
6072: 4E                   DFB  #<PSHAPE3
6073: 66                   DFB  #>PSHAPE3
6074: 5D                   DFB  #<PSHAPE4
6075: 66                   DFB  #>PSHAPE4
6076: 6C                   DFB  #<PSHAPE5
6077: 66                   DFB  #>PSHAPE5
6078: 7B                   DFB  #<PSHAPE6
6079: 66                   DFB  #>PSHAPE6
607A: 8A                   DFB  #<PSHAPE7
607B: 66                   DFB  #>PSHAPE7
607C: AD 50 C0    PGM      LDA  GRAPHICS   ;HIRES,P.1
607F: AD 52 C0             LDA  MIXOFF
6082: AD 57 C0             LDA  HIRES
6085: AD 54 C0             LDA  PAGE1
6088: A9 00                LDA  #$00       ;CLEAR SCREEN 1
608A: 85 1A                STA  LOW
608C: A9 20                LDA  #$20
608E: 85 1B                STA  HIGH
6090: A0 00       CLR1     LDY  #$00
6092: A9 00                LDA  #$00
6094: 91 1A       CLR      STA  (LOW),Y
6096: C8                   INY
6097: D0 FB                BNE  CLR
6099: E6 1B                INC  HIGH
609B: A5 1B                LDA  HIGH
```

```
609D: C9 40              CMP  #$40
609F: 90 EF              BLT  CLR1
60A1: A9 50              LDA  #$50        ;LOAD DELAY
60A3: 8D 0D 60           STA  DELAY
60A6: A2 B7              LDX  #$B7        ;DRAW BOTTOM LINE
60A8: A0 00              LDY  #$00
60AA: BD 9F 68           LDA  HI,X
60AD: 85 1B              STA  HIGH
60AF: BD 5F 69           LDA  LO,X
60B2: 85 1A              STA  LOW
60B4: A9 7F              LDA  #$7F
60B6: 91 1A      LN      STA  (LOW),Y
60B8: C8                 INY
60B9: C0 27              CPY  #$27
60BB: 90 F9              BLT  LN
60BD: A9 00              LDA  #$00
60BF: 8D 3C 60           STA  PCOUNTER
60C2: 8D 51 60           STA  ST
                 ********** MAIN PROGRAM **********
60C5: 20 70 61           JSR  SINITIAL    ;INITIALIZATION
60C8: 20 56 61           JSR  MINITIAL
60CB: 20 84 61   PI      JSR  PINITIAL
60CE: 20 65 61   BI      JSR  BINITIAL
60D1: 20 C6 61   PADDLE  JSR  PDLE        ;READ PADDLE
60D4: 20 01 62           JSR  MDRAW       ;DRAW MAN
60D7: EE 41 60   PSTART  INC  PSHPNO      ;FIRST SHAPE NUMBER TO ZERO
60DA: AD 41 60           LDA  PSHPNO
60DD: C9 07              CMP  #$07        ;DRAWN ALL 7 SHAPES?
60DF: 90 1B              BLT  PSTART2     ;IF NO, DRAW PLANE
60E1: EE 3F 60           INC  PBYTE       ;IF YES, NEXT SCREEN BYTE
60E4: AD 3F 60           LDA  PBYTE
60E7: C9 26              CMP  #$26        ;END OF SCREEN?
60E9: 90 09              BLT  PSTART1     ;IF NO, RESET SHAPE NO. &
                 *                         CONTINUE DRAW
60EB: 20 84 61           JSR  PINITIAL    ;IF YES, INITIALIZE PLANE AND
60EE: 20 01 62           JSR  MDRAW           ERASE MAN AND
60F1: 4C D1 60           JMP  PADDLE              GO BACK TO PADDLE READ
60F4: A9 FF      PSTART1 LDA  #$FF
60F6: 8D 41 60           STA  PSHPNO
60F9: 4C D7 60           JMP  PSTART
60FC: 20 AA 61   PSTART2 JSR  PLOADSHP
60FF: 20 4B 62           JSR  PDRAW       ;DRAW PLANE
6102: EE 3B 60           INC  DE          ;ACCESS SPEAKER EVERY OTHER CYCLE
6105: AD 3B 60           LDA  DE
                 **
6108: 4A                 LSR              ;C=0 IF DE IS EVEN
6109: 90 03              BCC  BUL         ;C=1 IF DE IS ODD
610B: 2C 30 C0           BIT  SPEAKER
610E: AD 0B 60   BUL     LDA  BULON
6111: C9 01              CMP  #$01        ;IS BULLET ON?
6113: F0 19              BEQ  BULLET      ;IF YES, CONTINUE BULLET DRAW
6115: AD 61 C0           LDA  BUTTON      ;IF NO, IS BUTTON PRESSED?
6118: 30 0F              BMI  BULLET1     ;IF YES, DRAW BULLET
611A: AD 0D 60           LDA  DELAY       ;IF NO,
611D: 20 A8 FC           JSR  WAIT            DELAY AND
6120: 20 01 62           JSR  MDRAW               ERASE MAN AND
6123: 20 4B 62           JSR  PDRAW                   ERASE PLANE
6126: 4C D1 60           JMP  PADDLE          READ PADDLE AGAIN
6129: A9 01      BULLET1 LDA  #$01        ;SET BULLET ON
612B: 8D 0B 60           STA  BULON
612E: 20 95 62   BULLET  JSR  LOADBUL     ;LOAD BULLET SHAPE INTO BTEMP
```

```
6131: 20 B8 62            JSR  BDRAW      ;DRAW BULLET & TEST FOR COLLISION
6134: AD 0D 60            LDA  DELAY
6137: 20 A8 FC            JSR  WAIT       ;DELAY
613A: 20 F8 62            JSR  BXDRAW     ;ERASE BULLET
613D: 20 01 62            JSR  MDRAW      ;ERASE MAN
6140: 20 4B 62            JSR  PDRAW      ;ERASE PLANE
6143: AD 05 60            LDA  BLINE
6146: 38                  SEC
6147: E9 08               SBC  #$08       ;MOVE BLINE UP 8 LINES
6149: 8D 05 60            STA  BLINE
614C: C9 05               CMP  #$05       ;LESS THAN 5 LINES FROM TOP?
614E: 90 03               BLT  LONG       ;IF YES, TAKE BRANCH
6150: 4C D1 60            JMP  PADDLE     ;IF NO, READ PADDLE AGAIN
6153: 4C CE 60   LONG     JMP  BI
                 ********** SUBROUTINES **********
6156: A9 AA      MINITIAL LDA  #$AA
6158: 8D 03 60            STA  MLINE
615B: 8D 04 60            STA  MLINEA
615E: 18                  CLC
615F: 69 0D               ADC  #$0D
6161: 8D 06 60            STA  DEPTH
6164: 60                  RTS
                 ***********************
6165: A9 00      BINITIAL LDA  #$00       ;BULON = 0 IF
6167: 8D 0B 60            STA  BULON         BULLET NOT ON SCREEN
616A: A9 A4               LDA  #$A4
616C: 8D 05 60            STA  BLINE
616F: 60                  RTS
                 **********************
6170: A9 00      SINITIAL LDA  #$00       ;SCORE DISPLAYS THREE 0'S
6172: 8D 39 60            STA  SUM
6175: 8D 3A 60            STA  COUNTER
6178: AA                  TAX
6179: A0 11               LDY  #$11
617B: 20 6D 64   PR       JSR  PRINT
617E: C8                  INY
617F: C0 14               CPY  #$14
6181: 90 F8               BLT  PR
6183: 60                  RTS
                 **********************
6184: A9 FF      PINITIAL LDA  #$FF       ;PSHPNO LOADED WITH #$FF SO FIRST
                 *                        INC PSHPNO WILL LOAD PSHPNO
                 *                        WITH ZERO
6186: 8D 41 60            STA  PSHPNO
6189: EE 3C 60            INC  PCOUNTER   ;PINITIAL AND PCOUNTER ACCESSED
                 *                        ONLY ON COLLISION OR
                 *                        END OF SCREEN
618C: AD 3C 60            LDA  PCOUNTER
618F: C9 65               CMP  #$65       ;PCOUNTER MORE THAN 100?
6191: 90 03               BLT  PCONT      ;IF NO, CONTINUE P INITIALIZATION
6193: 4C 62 64            JMP  STOP2      ;IF YES, STOP GAME
6196: A9 00      PCONT    LDA  #$00
6198: 8D 3F 60            STA  PBYTE
619B: A9 08               LDA  #$08
619D: 8D 3E 60            STA  PLINEA
61A0: 8D 3D 60            STA  PLINE
61A3: 18                  CLC
61A4: 69 05               ADC  #$05
61A6: 8D 40 60            STA  PDEPTH
61A9: 60                  RTS
                 ************************
```

```
61AA: AD 41 60   PLOADSHP LDA  PSHPNO
61AD: 0A                  ASL
61AE: AA                  TAX
61AF: BD 6E 60            LDA  PSHPADR,X
61B2: 85 1A               STA  LOW
61B4: BD 6F 60            LDA  PSHPADR+1,X
61B7: 85 1B               STA  HIGH
61B9: A0 00               LDY  #$00
61BB: B1 1A      PLOADSHP1 LDA (LOW),Y
61BD: 99 42 60            STA  PTEMP,Y
61C0: C8                  INY
61C1: C0 0F               CPY  #$0F
61C3: 90 F6               BLT  PLOADSHP1
61C5: 60                  RTS
                 *************************
61C6: A2 00      PDLE     LDX  #$00
61C8: 20 1E FB            JSR  PREAD       ;READ PADDLE 0
61CB: 98                  TYA
61CC: 8D 07 60            STA  MHORIZ      ;0-255 IN MHORIZ
61CF: AD 0B 60            LDA  BULON
61D2: C9 01               CMP  #$01        ;IS BULLET ON?
61D4: F0 06               BEQ  PDLE1       ;IF YES, TAKE BRANCH
61D6: AD 07 60            LDA  MHORIZ      ;IF NO, SET BHORIZ EQUAL
61D9: 8D 08 60            STA  BHORIZ           TO MHORIZ
61DC: AC 07 60   PDLE1    LDY  MHORIZ
61DF: B9 99 66            LDA  BYTETBL,Y   ;CONVERT 0-255 TO 0-36 (BYTE)
61E2: 8D 0A 60            STA  HORIZM      ;MAN BYTE POSITION
61E5: B9 9C 67            LDA  OFFSET,Y    ;GET SHAPE NUMBER
61E8: 0A                  ASL              ;LOAD SHAPE INTO MTEMP
61E9: AA                  TAX
61EA: BD 52 60            LDA  MSHPADR,X
61ED: 85 1A               STA  LOW
61EF: BD 53 60            LDA  MSHPADR+1,X
61F2: 85 1B               STA  HIGH
61F4: A0 00               LDY  #$00
61F6: B1 1A      LOAD     LDA  (LOW),Y
61F8: 99 0F 60            STA  MTEMP,Y
61FB: C8                  INY
61FC: C0 27               CPY  #$27
61FE: 90 F6               BLT  LOAD
6200: 60                  RTS
                 ***********************
6201: A9 00      MDRAW    LDA  #$00
6203: 8D 0C 60            STA  XCOUNT
6206: AE 03 60   MDRAW1   LDX  MLINE
6209: AC 0A 60            LDY  HORIZM
620C: BD 9F 68            LDA  HI,X
620F: 85 1B               STA  HIGH
6211: BD 5F 69            LDA  LO,X
6214: 85 1A               STA  LOW
6216: AE 0C 60            LDX  XCOUNT
6219: B1 1A               LDA  (LOW),Y
621B: 5D 0F 60            EOR  MTEMP,X
621E: 91 1A               STA  (LOW),Y
6220: C8                  INY
6221: B1 1A               LDA  (LOW),Y
6223: 5D 10 60            EOR  MTEMP+1,X
6226: 91 1A               STA  (LOW),Y
6228: C8                  INY
6229: B1 1A               LDA  (LOW),Y
622B: 5D 11 60            EOR  MTEMP+2,X
```

```
622E: 91 1A              STA  (LOW),Y
6230: EE 0C 60           INC  XCOUNT
6233: EE 0C 60           INC  XCOUNT
6236: EE 0C 60           INC  XCOUNT
6239: EE 03 60           INC  MLINE
623C: AD 03 60           LDA  MLINE
623F: CD 06 60           CMP  DEPTH
6242: 90 C2              BLT  MDRAW1
6244: AD 04 60           LDA  MLINEA     ;RESET LINE
6247: 8D 03 60           STA  MLINE
624A: 60                 RTS
                **********************
624B: A9 00     PDRAW    LDA  #$00
624D: 8D 0C 60           STA  XCOUNT
6250: AC 3F 60  PDRAW1   LDY  PBYTE
6253: AE 3D 60           LDX  PLINE
6256: BD 9F 68           LDA  HI,X
6259: 85 1B              STA  HIGH
625B: BD 5F 69           LDA  LO,X
625E: 85 1A              STA  LOW
6260: AE 0C 60           LDX  XCOUNT
6263: B1 1A              LDA  (LOW),Y
6265: 5D 42 60           EOR  PTEMP,X
6268: 91 1A              STA  (LOW),Y
626A: C8                 INY
626B: B1 1A              LDA  (LOW),Y
626D: 5D 43 60           EOR  PTEMP+1,X
6270: 91 1A              STA  (LOW),Y
6272: C8                 INY
6273: B1 1A              LDA  (LOW),Y
6275: 5D 44 60           EOR  PTEMP+2,X
6278: 91 1A              STA  (LOW),Y
627A: EE 0C 60           INC  XCOUNT
627D: EE 0C 60           INC  XCOUNT
6280: EE 0C 60           INC  XCOUNT
6283: EE 3D 60           INC  PLINE
6286: AD 3D 60           LDA  PLINE
6289: CD 40 60           CMP  PDEPTH
628C: 90 C2              BLT  PDRAW1
628E: AD 3E 60           LDA  PLINEA     ;RESET LINE
6291: 8D 3D 60           STA  PLINE
6294: 60                 RTS
                *************************
6295: AC 08 60  LOADBUL  LDY  BHORIZ     ;CONVERTS 0-255 TO
6298: B9 99 66           LDA  BYTETBL,Y     SCREEN BYTE (0-36)
629B: 18                 CLC             ;ADD 2 TO ALIGN BULLET
629C: 69 02              ADC  #$02          WITH GUN
629E: 8D 09 60           STA  HORIZB     ;BULLET BYTE POSITION
62A1: B9 9C 67           LDA  OFFSET,Y   ;GET BULLET SHAPE NUMBER
62A4: 0A                 ASL             ;LOAD BULLET SHAPE INTO BTEMP
62A5: AA                 TAX
62A6: BD 60 60           LDA  BSHPADR,X
62A9: 85 1A              STA  LOW
62AB: BD 61 60           LDA  BSHPADR+1,X
62AE: 85 1B              STA  HIGH
62B0: A0 00              LDY  #$00
62B2: B1 1A              LDA  (LOW),Y
62B4: 8D 0E 60           STA  BTEMP
62B7: 60                 RTS
                **********************
```

```
62B8: AE 05 60   BDRAW     LDX  BLINE
62BB: AC 09 60             LDY  HORIZB
62BE: BD 9F 68             LDA  HI,X
62C1: 85 1B                STA  HIGH
62C3: BD 5F 69             LDA  LO,X
62C6: 85 1A                STA  LOW
62C8: B1 1A                LDA  (LOW),Y
62CA: 2D 0E 60             AND  BTEMP      ;RESULT IS 0 IF NO COLLISION
62CD: C9 00                CMP  #$00
62CF: F0 03                BEQ  NOHIT
62D1: 4C DC 62             JMP  COLLISION
62D4: B1 1A      NOHIT     LDA  (LOW),Y    ;DRAW BULLET
62D6: 4D 0E 60             EOR  BTEMP
62D9: 91 1A                STA  (LOW),Y
62DB: 60                   RTS
                 ***********************
62DC: 20 4B 62   COLLISION JSR  PDRAW      ;ERASE PLANE
62DF: EE 39 60             INC  SUM        ;ADD 1 TO SCORE
62E2: 20 1D 64             JSR  SCORE      ;DISPLAY SCORE
62E5: 20 10 63             JSR  EXPLODE    ;EXPLOSION DISPLAY AND SOUND
62E8: AD 51 60             LDA  ST         ;IF COUNT=100,
62EB: C9 01                CMP  #$01       THEN GO TO
62ED: F0 06                BEQ  LG         STOP PROGRAM
62EF: 20 01 62             JSR  MDRAW      ;ERASE MAN
62F2: 4C CB 60             JMP  PI         ;INITIALIZE P, B, AND READ PADDLE
62F5: 4C 62 64   LG        JMP  STOP2
                 ***********************
62F8: AE 05 60   BXDRAW    LDX  BLINE      ;BDRAW WITHOUT COLLISION TEST
62FB: AC 09 60             LDY  HORIZB
62FE: BD 9F 68             LDA  HI,X
6301: 85 1B                STA  HIGH
6303: BD 5F 69             LDA  LO,X
6306: 85 1A                STA  LOW
6308: B1 1A                LDA  (LOW),Y
630A: 4D 0E 60             EOR  BTEMP
630D: 91 1A                STA  (LOW),Y
630F: 60                   RTS
                 ***********************
6310: 20 CD 63   EXPLODE   JSR  INITE1
6313: 20 61 63             JSR  DRAWE1     ;DRAW
6316: 20 53 63             JSR  SOUND      ;EXPLOSION SOUND
6319: 20 CD 63             JSR  INITE1
631C: 20 61 63             JSR  DRAWE1     ;ERASE
631F: 20 E1 63             JSR  INITE2
6322: 20 61 63             JSR  DRAWE1     ;DRAW
6325: A9 BB                LDA  #$BB
6327: 20 A8 FC             JSR  WAIT
632A: 20 E1 63             JSR  INITE2
632D: 20 61 63             JSR  DRAWE1     ;ERASE
6330: 20 F5 63             JSR  INITE3
6333: 20 61 63             JSR  DRAWE1     ;DRAW
6336: A9 BB                LDA  #$BB
6338: 20 A8 FC             JSR  WAIT
633B: 20 F5 63             JSR  INITE3
633E: 20 61 63             JSR  DRAWE1     ;ERASE
6341: 20 09 64             JSR  INITE4
6344: 20 90 63             JSR  DRAWE2     ;DRAW
6347: A9 FF                LDA  #$FF
6349: 20 A8 FC             JSR  WAIT
```

```
634C: 20 09 64              JSR  INITE4
634F: 20 90 63              JSR  DRAWE2      ;ERASE
6352: 60                    RTS
                 ************************
6353: A0 02      SOUND      LDY  #$02        ;EXPLOSION SOUND
6355: 2C 30 C0   SOUND1     BIT  SPEAKER
6358: A9 60                 LDA  #$60
635A: 20 A8 FC              JSR  WAIT
635D: 88                    DEY
635E: D0 F5                 BNE  SOUND1
6360: 60                    RTS
                 ************************
6361: AC 09 60   DRAWE1     LDY  HORIZB      ;ROUTINE FOR FIRST 3
6364: AE 36 60              LDX  ELINE            EXPLOSION SHAPES
6367: BD 9F 68              LDA  HI,X
636A: 85 1B                 STA  HIGH
636C: BD 5F 69              LDA  LO,X
636F: 85 1A                 STA  LOW
6371: AE 0C 60              LDX  XCOUNT
6374: B1 1A                 LDA  (LOW),Y
6376: 5D 06 66              EOR  ESHAPE,X
6379: 91 1A                 STA  (LOW),Y
637B: EE 0C 60              INC  XCOUNT
637E: EE 36 60              INC  ELINE
6381: AD 36 60              LDA  ELINE
6384: CD 38 60              CMP  EDEPTH
6387: 90 D8                 BLT  DRAWE1
6389: AD 37 60              LDA  ELINEA
638C: 8D 36 60              STA  ELINE
638F: 60                    RTS
                 ************************
6390: AC 09 60   DRAWE2     LDY  HORIZB      ;ROUTINE FOR FOURTH
6393: AE 36 60              LDX  ELINE            EXPLOSION SHAPE
6396: BD 9F 68              LDA  HI,X
6399: 85 1B                 STA  HIGH
639B: BD 5F 69              LDA  LO,X
639E: 85 1A                 STA  LOW
63A0: AE 0C 60              LDX  XCOUNT
63A3: B1 1A                 LDA  (LOW),Y
63A5: 5D 06 66              EOR  ESHAPE,X
63A8: 91 1A                 STA  (LOW),Y
63AA: EE 0C 60              INC  XCOUNT
63AD: C8                    INY
63AE: AE 0C 60              LDX  XCOUNT
63B1: B1 1A                 LDA  (LOW),Y
63B3: 5D 06 66              EOR  ESHAPE,X
63B6: 91 1A                 STA  (LOW),Y
63B8: EE 0C 60              INC  XCOUNT
63BB: EE 36 60              INC  ELINE
63BE: AD 36 60              LDA  ELINE
63C1: CD 38 60              CMP  EDEPTH
63C4: 90 CA                 BLT  DRAWE2
63C6: AD 37 60              LDA  ELINEA
63C9: 8D 36 60              STA  ELINE
63CC: 60                    RTS
                 ************************
63CD: A9 00      INITE1     LDA  #$00        ;INITIALIZE FIRST EXPLOSION
```

```
63CF: 8D 0C 60  459            STA  XCOUNT
63D2: A9 09     460            LDA  #$09
63D4: 8D 37 60  461            STA  ELINEA
63D7: 8D 36 60  462            STA  ELINE
63DA: 18        463            CLC
63DB: 69 05     464            ADC  #$05
63DD: 8D 38 60  465            STA  EDEPTH
63E0: 60        466            RTS
63E1: A9 05     467   INITE2   LDA  #$05        ;INITIALIZE SECOND EXPLOSION
63E3: 8D 0C 60  468            STA  XCOUNT
63E6: A9 09     469            LDA  #$09
63E8: 8D 37 60  470            STA  ELINEA
63EB: 8D 36 60  471            STA  ELINE
63EE: 18        472            CLC
63EF: 69 05     473            ADC  #$05
63F1: 8D 38 60  474            STA  EDEPTH
63F4: 60        475            RTS
63F5: A9 0A     476   INITE3   LDA  #$0A        ;INITIALIZE THIRD EXPLOSION
63F7: 8D 0C 60  477            STA  XCOUNT
63FA: A9 05     478            LDA  #$05
63FC: 8D 37 60  479            STA  ELINEA
63FF: 8D 36 60  480            STA  ELINE
6402: 18        481            CLC
6403: 69 08     482            ADC  #$08
6405: 8D 38 60  483            STA  EDEPTH
6408: 60        484            RTS
6409: A9 12     485   INITE4   LDA  #$12        ;INITIALIZE FOURTH EXPLOSION
640B: 8D 0C 60  486            STA  XCOUNT
640E: A9 01     487            LDA  #$01
6410: 8D 37 60  488            STA  ELINEA
6413: 8D 36 60  489            STA  ELINE
6416: 18        490            CLC
6417: 69 0C     491            ADC  #$0C
6419: 8D 38 60  492            STA  EDEPTH
641C: 60        493            RTS
                494   ************************
641D: AD 39 60  495   SCORE    LDA  SUM         ;GET SCORE (0-9)
6420: C9 0A     496            CMP  #$0A        ;GREATER THAN 9?
6422: B0 0A     497            BGE  C10         ;IF YES, BRANCH
6424: 0A        498            ASL              ;IF NO, MULTIPLY BY 8
6425: 0A        499            ASL
6426: 0A        500            ASL
6427: AA        501            TAX
6428: A0 13     502            LDY  #$13        ;BYTE POSITION FOR FIRST DIGIT
642A: 20 6D 64  503            JSR  PRINT       ;PRINT DIGIT
642D: 60        504            RTS
642E: EE 3A 60  505   C10      INC  COUNTER     ;INC COUNTER (INITIALLY 0)
6431: AD 3A 60  506            LDA  COUNTER
6434: C9 0A     507            CMP  #$0A        ;MORE THAN 9?
6436: B0 11     508            BGE  STOP1       ;IF YES, PRINT 100 AND STOP GAME
6438: 0A        509            ASL              ;IF NO, MULTIPLY BY 8
6439: 0A        510            ASL
643A: 0A        511            ASL
643B: AA        512            TAX
643C: A0 12     513            LDY  #$12        ;BYTE POSITION OF MIDDLE DIGIT
643E: 20 6D 64  514            JSR  PRINT       ;PRINT DIGIT
6441: A9 00     515            LDA  #$00        ;ZERO SUM AND
```

```
6443: 8D 39 60               STA  SUM          RETURN TO PRINT 0
6446: 4C 1D 64               JMP  SCORE          IN FIRST DIGIT POSITION
6449: A2 08         STOP1    LDX  #$08         ;PRINT 100 IN COUNTER
644B: A0 11                  LDY  #$11
644D: 20 6D 64               JSR  PRINT
6450: A2 00                  LDX  #$00
6452: A0 12                  LDY  #$12
6454: 20 6D 64               JSR  PRINT
6457: A0 13                  LDY  #$13
6459: 20 6D 64               JSR  PRINT
645C: A9 01                  LDA  #$01         ;SET ST TO INDICATE
645E: 8D 51 60               STA  ST           COUNTER=100
6461: 60                     RTS
6462: 2C 00 C0      STOP2    BIT  $C000        ;ANY KEY PRESSED?
6465: 10 FB                  BPL  STOP2        ;IF NO, BRANCH BACK & WAIT
                    *                          FOR KEYSTROKE
6467: 2C 10 C0               BIT  $C010        ;IF YES, CLEAR KEYBOARD STROBE
646A: 4C 7C 60               JMP  PGM           AND START PROGRAM OVER
                    **************************
646D: BD 9E 64      PRINT    LDA  NSHAPE,X     ;RETRIEVE NUMBER SHAPE
6470: 99 D0 23               STA  $23D0,Y      ;LINE #$B8 (184)
6473: BD 9F 64               LDA  NSHAPE+1,X
6476: 99 D0 27               STA  $27D0,Y      ;LINE #$B9 (185)
6479: BD A0 64               LDA  NSHAPE+2,X
647C: 99 D0 2B               STA  $2BD0,Y      ;LINE #$BA (186)
647F: BD A1 64               LDA  NSHAPE+3,X
6482: 99 D0 2F               STA  $2FD0,Y      ;LINE #$BB (187)
6485: BD A2 64               LDA  NSHAPE+4,X
6488: 99 D0 33               STA  $33D0,Y      ;LINE #$BC (188)
648B: BD A3 64               LDA  NSHAPE+5,X
648E: 99 D0 37               STA  $37D0,Y      ;LINE #$BD (189)
6491: BD A4 64               LDA  NSHAPE+6,X
6494: 99 D0 3B               STA  $3BD0,Y      ;LINE #$BE (190)
6497: BD A5 64               LDA  NSHAPE+7,X
649A: 99 D0 3F               STA  $3FD0,Y      ;LINE #$BF (191)
649D: 60                     RTS
                    ****************************
649E: 00 1C 22      NSHAPE   HEX  001C22222222221C     ;NUMBER SHAPES - "0"
64A1: 22 22 22 22 1C
64A6: 00 08 0C               HEX  00080C080808081C     ;"1"
64A9: 08 08 08 08 1C
64AE: 00 1C 22               HEX  001C22201008043E     ;"2"
64B1: 20 10 08 04 3E
64B6: 00 1C 22               HEX  001C22201C20221C     ;"3"
64B9: 20 1C 20 22 1C
64BE: 00 10 18               HEX  00101814123E1010     ;"4"
64C1: 14 12 3E 10 10
64C6: 00 3E 02               HEX  003E021E2020201E     ;"5"
64C9: 1E 20 20 20 1E
64CE: 00 1C 22               HEX  001C22021E22221C     ;"6"
64D1: 02 1E 22 22 1C
64D6: 00 3E 20               HEX  003E201008040404     ;"7"
64D9: 10 08 04 04 04
64DE: 00 1C 22               HEX  001C22221C22221C     ;"8"
64E1: 22 1C 22 22 1C
64E6: 00 1C 22               HEX  001C22223C20221C     ;"9"
64E9: 22 3C 20 22 1C
```

```
64EE: 00 0E 01   563  MSHAPE1  HEX   000E01000E01000E01    ;MAN SHAPE TABLES
64F1: 00 0E 01 00 0E 01
64F7: 00 44 01   564           HEX   004401007F00601F00
64FA: 00 7F 00 60 1F 00
6500: 30 1F 00   565           HEX   301F00181F00001F00
6503: 18 1F 00 00 1F 00
6509: 00 1F 00   566           HEX   001F00001B00403100
650C: 00 1B 00 40 31 00
6512: 60 60 00   567           HEX   606000
6515: 00 1C 02   568  MSHAPE2  HEX   001C02001C02001C02
6518: 00 1C 02 00 1C 02
651E: 00 08 03   569           HEX   000803007E01003E00
6521: 00 7E 01 00 3E 00
6527: 00 3F 00   570           HEX   003F00403F00003E00
652A: 40 3F 00 00 3E 00
6530: 00 3E 00   571           HEX   003E00003600003600
6533: 00 36 00 00 36 00
6539: 00 63 00   572           HEX   006300
653C: 00 38 04   573  MSHAPE3  HEX   003804003804003804
653F: 00 38 04 00 38 04
6545: 00 10 06   574           HEX   001006007C03007C00
6548: 00 7C 03 00 7C 00
654E: 00 7C 00   575           HEX   007C00007E00007C00
6551: 00 7E 00 00 7C 00
6557: 00 38 00   576           HEX   003800003800006C00
655A: 00 38 00 00 6C 00
6560: 00 46 01   577           HEX   004601
6563: 00 70 08   578  MSHAPE4  HEX   007008007008007008
6566: 00 70 08 00 70 08
656C: 00 20 0C   579           HEX   00200C007807007801
656F: 00 78 07 00 78 01
6575: 00 78 01   580           HEX   007801007801007801
6578: 00 78 01 00 78 01
657E: 00 70 00   581           HEX   007000007000007000
6581: 00 70 00 00 70 00
6587: 00 70 00   582           HEX   007000
658A: 00 60 11   583  MSHAPE5  HEX   006011006011006011
658D: 00 60 11 00 60 11
6593: 00 40 18   584           HEX   00401800700F007003
6596: 00 70 0F 00 70 03
659C: 00 70 03   585           HEX   007003007803007003
659F: 00 78 03 00 70 03
65A5: 00 60 01   586           HEX   006001006001003003
65A8: 00 60 01 00 30 03
65AE: 00 18 06   587           HEX   001806
65B1: 00 40 23   588  MSHAPE6  HEX   004023004023004023
65B4: 00 40 23 00 40 23
65BA: 00 00 31   589           HEX   00003100601F006007
65BD: 00 60 1F 00 60 07
65C3: 00 70 07   590           HEX   007007007807006007
65C6: 00 78 07 00 60 07
65CC: 00 60 07   591           HEX   006007006006006006
65CF: 00 60 06 00 60 06
65D5: 00 30 0C   592           HEX   00300C
65D8: 00 00 47   593  MSHAPE7  HEX   000047000047000047
65DB: 00 00 47 00 00 47
65E1: 00 00 62   594           HEX   00006200403F00700F
```

```
65E4: 00 40 3F 00 70 0F
65EA: 00 58 0F  595             HEX  00580F004C0F00400F
65ED: 00 4C 0F 00 40 0F
65F3: 00 40 0F  596             HEX  00400F00400D006018
65F6: 00 40 0D 00 60 18
65FC: 00 30 30  597             HEX  003030
65FF: 01        598  BSHAPE1    HEX  01                 ;BULLET SHAPES
6600: 02        599  BSHAPE2    HEX  02
6601: 04        600  BSHAPE3    HEX  04
6602: 08        601  BSHAPE4    HEX  08
6603: 10        602  BSHAPE5    HEX  10
6604: 20        603  BSHAPE6    HEX  20
6605: 40        604  BSHAPE7    HEX  40
6606: 28 22 1A  605  ESHAPE     HEX  28221A2514         ;EXPLOSION SHAPES - NO. 1
6609: 25 14
660B: 2C 52 44  606             HEX  2C5244320C             ;NO. 2
660E: 32 0C
6610: 38 3E 7F  607             HEX  383E7F7E7E3F3F1C       ;NO. 3
6613: 7E 7E 3F 3F 1C
6618: 18 06 7C  608             HEX  18067C0F7C3F7E3F       ;NO. 4
661B: 0F 7C 3F 7E 3F
6620: 7C 7F 7C  609             HEX  7C7F7C3F7E3F7F1F
6623: 3F 7E 3F 7F 1F
6628: 7E 0F 7C  610             HEX  7E0F7C1F700F4003
662B: 1F 70 0F 40 03
6630: 02 00 00  611  PSHAPE1    HEX  0200000600007E1F00     ;PLANE SHAPES
6633: 06 00 00 7E 1F 00
6639: 7E 37 00  612             HEX  7E37007E7F00
663C: 7E 7F 00
663F: 04 00 00  613  PSHAPE2    HEX  0400000C00007C3F00
6642: 0C 00 00 7C 3F 00
6648: 7C 6F 00  614             HEX  7C6F007C7F01
664B: 7C 7F 01
664E: 08 00 00  615  PSHAPE3    HEX  080000180000787F00
6651: 18 00 00 78 7F 00
6657: 78 5F 01  616             HEX  785F01787F03
665A: 78 7F 03
665D: 10 00 00  617  PSHAPE4    HEX  100000300000707F01
6660: 30 00 00 70 7F 01
6666: 70 3F 03  618             HEX  703F03707F07
6669: 70 7F 07
666C: 20 00 00  619  PSHAPE5    HEX  200000600000607F03
666F: 60 00 00 60 7F 03
6675: 60 7F 06  620             HEX  607F06607F0F
6678: 60 7F 0F
667B: 40 00 00  621  PSHAPE6    HEX  400000400100407F07
667E: 40 01 00 40 7F 07
6684: 40 7F 0D  622             HEX  407F0D407F1F
6687: 40 7F 1F
668A: 00 01 00  623  PSHAPE7    HEX  000100000300007F0F
668D: 00 03 00 00 7F 0F
6693: 00 7F 1B  624             HEX  007F1B007F3F
6696: 00 7F 3F
6699: 00 00 00  625  BYTETBL    HEX  00000000000000
669C: 00 00 00 00
66A0: 01 01 01  626             HEX  01010101010101
66A3: 01 01 01 01
```

```
66A7: 02 02 02          HEX  02020202020202
66AA: 02 02 02 02
66AE: 03 03 03          HEX  03030303030303
66B1: 03 03 03 03
66B5: 04 04 04          HEX  04040404040404
66B8: 04 04 04 04
66BC: 05 05 05          HEX  05050505050505
66BF: 05 05 05 05
66C3: 06 06 06          HEX  06060606060606
66C6: 06 06 06 06
66CA: 07 07 07          HEX  07070707070707
66CD: 07 07 07 07
66D1: 08 08 08          HEX  08080808080808
66D4: 08 08 08 08
66D8: 09 09 09          HEX  09090909090909
66DB: 09 09 09 09
66DF: 0A 0A 0A          HEX  0A0A0A0A0A0A0A
66E2: 0A 0A 0A 0A
66E6: 0B 0B 0B          HEX  0B0B0B0B0B0B0B
66E9: 0B 0B 0B 0B
66ED: 0C 0C 0C          HEX  0C0C0C0C0C0C0C
66F0: 0C 0C 0C 0C
66F4: 0D 0D 0D          HEX  0D0D0D0D0D0D0D
66F7: 0D 0D 0D 0D
66FB: 0E 0E 0E          HEX  0E0E0E0E0E0E0E
66FE: 0E 0E 0E 0E
6702: 0F 0F 0F          HEX  0F0F0F0F0F0F0F
6705: 0F 0F 0F 0F
6709: 10 10 10          HEX  10101010101010
670C: 10 10 10 10
6710: 11 11 11          HEX  11111111111111
6713: 11 11 11 11
6717: 12 12 12          HEX  12121212121212
671A: 12 12 12 12
671E: 13 13 13          HEX  13131313131313
6721: 13 13 13 13
6725: 14 14 14          HEX  14141414141414
6728: 14 14 14 14
672C: 15 15 15          HEX  15151515151515
672F: 15 15 15 15
6733: 16 16 16          HEX  16161616161616
6736: 16 16 16 16
673A: 17 17 17          HEX  17171717171717
673D: 17 17 17 17
6741: 18 18 18          HEX  18181818181818
6744: 18 18 18 18
6748: 19 19 19          HEX  19191919191919
674B: 19 19 19 19
674F: 1A 1A 1A          HEX  1A1A1A1A1A1A1A
6752: 1A 1A 1A 1A
6756: 1B 1B 1B          HEX  1B1B1B1B1B1B1B
6759: 1B 1B 1B 1B
675D: 1C 1C 1C          HEX  1C1C1C1C1C1C1C
6760: 1C 1C 1C 1C
6764: 1D 1D 1D          HEX  1D1D1D1D1D1D1D
6767: 1D 1D 1D 1D
676B: 1E 1E 1E          HEX  1E1E1E1E1E1E1E
676E: 1E 1E 1E 1E
6772: 1F 1F 1F          HEX  1F1F1F1F1F1F1F
```

```
6775: 1F 1F 1F 1F
6779: 20 20 20  657          HEX  20202020202020
677C: 20 20 20 20
6780: 21 21 21  658          HEX  21212121212121
6783: 21 21 21 21
6787: 22 22 22  659          HEX  22222222222222
678A: 22 22 22 22
678E: 23 23 23  660          HEX  23232323232323
6791: 23 23 23 23
6795: 24 24 24  661          HEX  24242424242424
6798: 24 24 24 24
679C: 00 01 02  662  OFFSET  HEX  00010203040506
679F: 03 04 05 06
67A3: 00 01 02  663          HEX  00010203040506
67A6: 03 04 05 06
67AA: 00 01 02  664          HEX  00010203040506
67AD: 03 04 05 06
67B1: 00 01 02  665          HEX  00010203040506
67B4: 03 04 05 06
67B8: 00 01 02  666          HEX  00010203040506
67BB: 03 04 05 06
67BF: 00 01 02  667          HEX  00010203040506
67C2: 03 04 05 06
67C6: 00 01 02  668          HEX  00010203040506
67C9: 03 04 05 06
67CD: 00 01 02  669          HEX  00010203040506
67D0: 03 04 05 06
67D4: 00 01 02  670          HEX  00010203040506
67D7: 03 04 05 06
67DB: 00 01 02  671          HEX  00010203040506
67DE: 03 04 05 06
67E2: 00 01 02  672          HEX  00010203040506
67E5: 03 04 05 06
67E9: 00 01 02  673          HEX  00010203040506
67EC: 03 04 05 06
67F0: 00 01 02  674          HEX  00010203040506
67F3: 03 04 05 06
67F7: 00 01 02  675          HEX  00010203040506
67FA: 03 04 05 06
67FE: 00 01 02  676          HEX  00010203040506
6801: 03 04 05 06
6805: 00 01 02  677          HEX  00010203040506
6808: 03 04 05 06
680C: 00 01 02  678          HEX  00010203040506
680F: 03 04 05 06
6813: 00 01 02  679          HEX  00010203040506
6816: 03 04 05 06
681A: 00 01 02  680          HEX  00010203040506
681D: 03 04 05 06
6821: 00 01 02  681          HEX  00010203040506
6824: 03 04 05 06
6828: 00 01 02  682          HEX  00010203040506
682B: 03 04 05 06
682F: 00 01 02  683          HEX  00010203040506
6832: 03 04 05 06
6836: 00 01 02  684          HEX  00010203040506
6839: 03 04 05 06
683D: 00 01 02  685          HEX  00010203040506
6840: 03 04 05 06
```

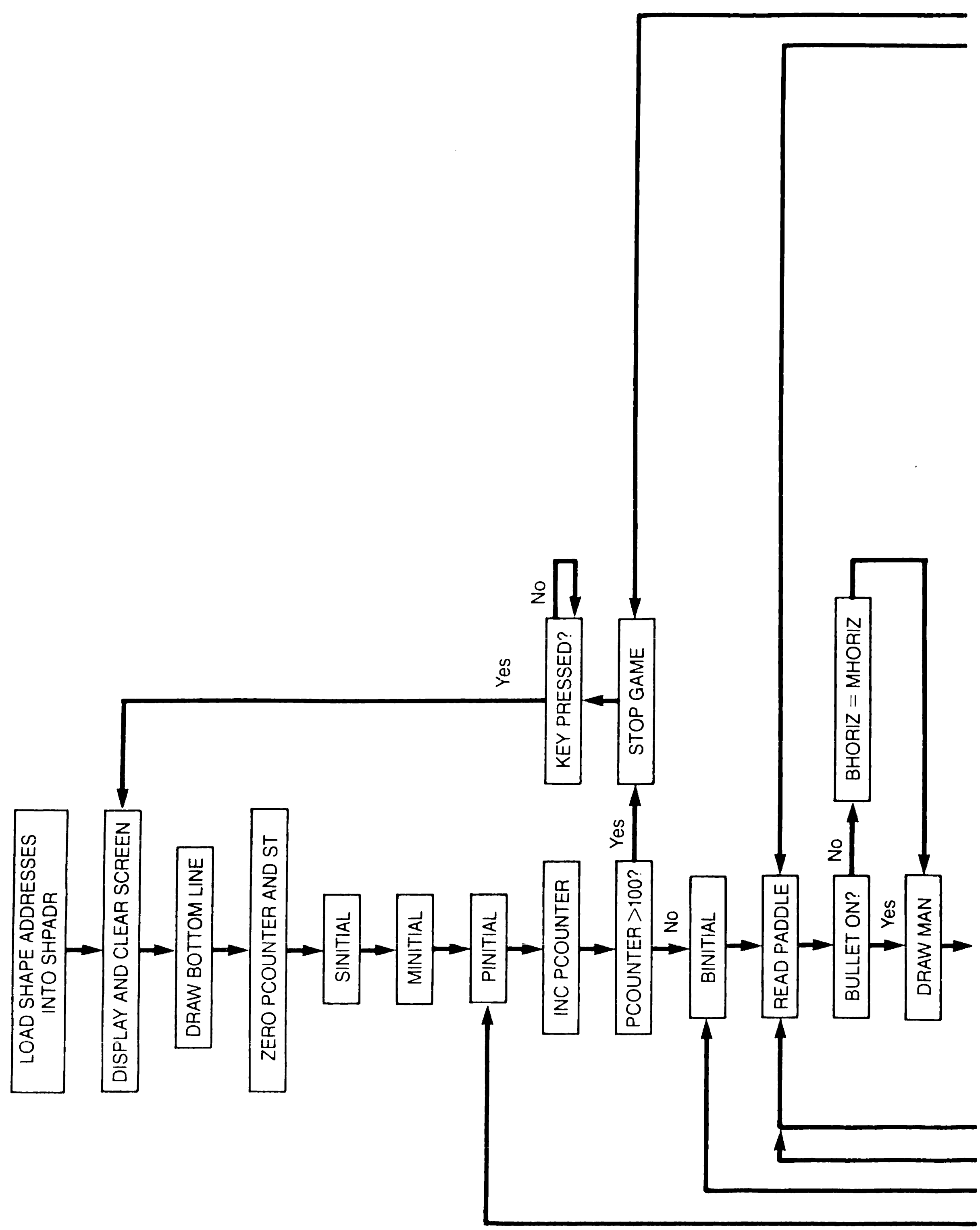
```
6844: 00 01 02   686            HEX  00010203040506
6847: 03 04 05 06
684B: 00 01 02   687            HEX  00010203040506
684E: 03 04 05 06
6852: 00 01 02   688            HEX  00010203040506
6855: 03 04 05 06
6859: 00 01 02   689            HEX  00010203040506
685C: 03 04 05 06
6860: 00 01 02   690            HEX  00010203040506
6863: 03 04 05 06
6867: 00 01 02   691            HEX  00010203040506
686A: 03 04 05 06
686E: 00 01 02   692            HEX  00010203040506
6871: 03 04 05 06
6875: 00 01 02   693            HEX  00010203040506
6878: 03 04 05 06
687C: 00 01 02   694            HEX  00010203040506
687F: 03 04 05 06
6883: 00 01 02   695            HEX  00010203040506
6886: 03 04 05 06
688A: 00 01 02   696            HEX  00010203040506
688D: 03 04 05 06
6891: 00 01 02   697            HEX  00010203040506
6894: 03 04 05 06
6898: 00 01 02   698            HEX  00010203040506
689B: 03 04 05 06
689F: 20 24 28   699  HI        HEX  2024282C3034383C ; HIGH BYTE LINE ADDRESSES
68A2: 2C 30 34 38 3C
68A7: 20 24 28   700            HEX  2024282C3034383C
68AA: 2C 30 34 38 3C
68AF: 21 25 29   701            HEX  2125292D3135393D
68B2: 2D 31 35 39 3D
68B7: 21 25 29   702            HEX  2125292D3135393D
68BA: 2D 31 35 39 3D
68BF: 22 26 2A   703            HEX  22262A2E32363A3E
68C2: 2E 32 36 3A 3E
68C7: 22 26 2A   704            HEX  22262A2E32363A3E
68CA: 2E 32 36 3A 3E
68CF: 23 27 2B   705            HEX  23272B2F33373B3F
68D2: 2F 33 37 3B 3F
68D7: 23 27 2B   706            HEX  23272B2F33373B3F
68DA: 2F 33 37 3B 3F
68DF: 20 24 28   707            HEX  2024282C3034383C
68E2: 2C 30 34 38 3C
68E7: 20 24 28   708            HEX  2024282C3034383C
68EA: 2C 30 34 38 3C
68EF: 21 25 29   709            HEX  2125292D3135393D
68F2: 2D 31 35 39 3D
68F7: 21 25 29   710            HEX  2125292D3135393D
68FA: 2D 31 35 39 3D
68FF: 22 26 2A   711            HEX  22262A2E32363A3E
6902: 2E 32 36 3A 3E
6907: 22 26 2A   712            HEX  22262A2E32363A3E
690A: 2E 32 36 3A 3E
690F: 23 27 2B   713            HEX  23272B2F33373B3F
6912: 2F 33 37 3B 3F
6917: 23 27 2B   714            HEX  23272B2F33373B3F
691A: 2F 33 37 3B 3F
691F: 20 24 28   715            HEX  2024282C3034383C
```

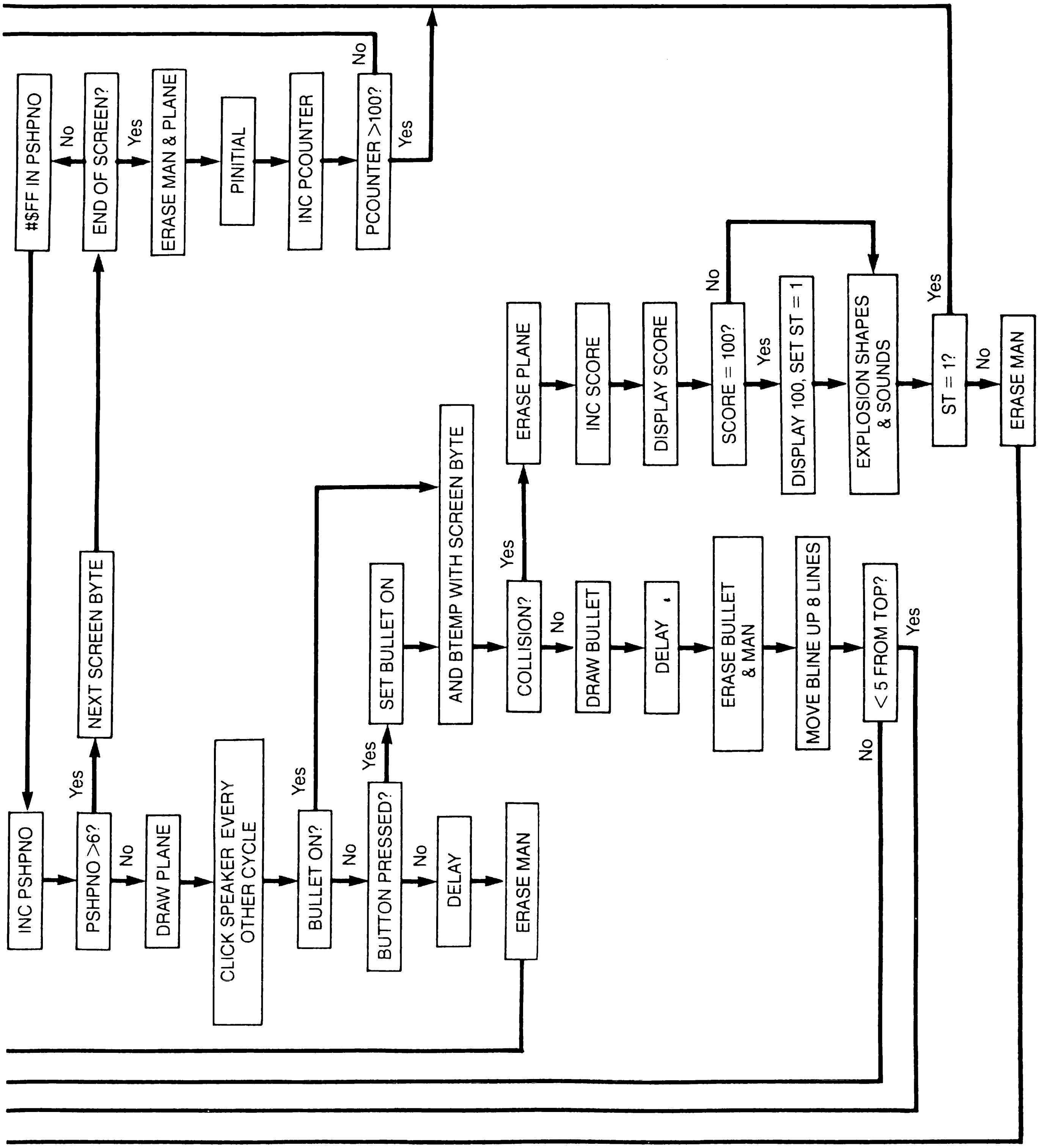
```
6922: 2C 30 34 38 3C
6927: 20 24 28   716             HEX  2024282C3034383C
692A: 2C 30 34 38 3C
692F: 21 25 29   717             HEX  2125292D3135393D
6932: 2D 31 35 39 3D
6937: 21 25 29   718             HEX  2125292D3135393D
693A: 2D 31 35 39 3D
693F: 22 26 2A   719             HEX  22262A2E32363A3E
6942: 2E 32 36 3A 3E
6947: 22 26 2A   720             HEX  22262A2E32363A3E
694A: 2E 32 36 3A 3E
694F: 23 27 2B   721             HEX  23272B2F33373B3F
6952: 2F 33 37 3B 3F
6957: 23 27 2B   722             HEX  23272B2F33373B3F
695A: 2F 33 37 3B 3F
695F: 00 00 00   723  LO         HEX  0000000000000000 ; LOW BYTE LINE ADDRESSES
6962: 00 00 00 00 00
6967: 80 80 80   724             HEX  8080808080808080
696A: 80 80 80 80 80
696F: 00 00 00   725             HEX  0000000000000000
6972: 00 00 00 00 00
6977: 80 80 80   726             HEX  8080808080808080
697A: 80 80 80 80 80
697F: 00 00 00   727             HEX  0000000000000000
6982: 00 00 00 00 00
6987: 80 80 80   728             HEX  8080808080808080
698A: 80 80 80 80 80
698F: 00 00 00   729             HEX  0000000000000000
6992: 00 00 00 00 00
6997: 80 80 80   730             HEX  8080808080808080
699A: 80 80 80 80 80
699F: 28 28 28   731             HEX  2828282828282828
69A2: 28 28 28 28 28
69A7: A8 A8 A8   732             HEX  A8A8A8A8A8A8A8A8
69AA: A8 A8 A8 A8 A8
69AF: 28 28 28   733             HEX  2828282828282828
69B2: 28 28 28 28 28
69B7: A8 A8 A8   734             HEX  A8A8A8A8A8A8A8A8
69BA: A8 A8 A8 A8 A8
69BF: 28 28 28   735             HEX  2828282828282828
69C2: 28 28 28 28 28
69C7: A8 A8 A8   736             HEX  A8A8A8A8A8A8A8A8
69CA: A8 A8 A8 A8 A8
69CF: 28 28 28   737             HEX  2828282828282828
69D2: 28 28 28 28 28
69D7: A8 A8 A8   738             HEX  A8A8A8A8A8A8A8A8
69DA: A8 A8 A8 A8 A8
69DF: 50 50 50   739             HEX  5050505050505050
69E2: 50 50 50 50 50
69E7: D0 D0 D0   740             HEX  D0D0D0D0D0D0D0D0
69EA: D0 D0 D0 D0 D0
69EF: 50 50 50   741             HEX  5050505050505050
69F2: 50 50 50 50 50
69F7: D0 D0 D0   742             HEX  D0D0D0D0D0D0D0D0
69FA: D0 D0 D0 D0 D0
69FF: 50 50 50   743             HEX  5050505050505050
6A02: 50 50 50 50 50
6A07: D0 D0 D0   744             HEX  D0D0D0D0D0D0D0D0
6A0A: D0 D0 D0 D0 D0
```

```
6A0F: 50 50 50  745          HEX  5050505050505050
6A12: 50 50 50 50 50
6A17: D0 D0 D0  746          HEX  D0D0D0D0D0D0D0D0
6A1A: D0 D0 D0 D0 D0


--End assembly--

2591 bytes


Symbol table - numerical order:

   LOW      =$1A      HIGH      =$1B      MLINE     =$6003    MLINEA   =$6004
   BLINE    =$6005    DEPTH     =$6006    MHORIZ    =$6007    BHORIZ   =$6008
   HORIZB   =$6009    HORIZM    =$600A    BULON     =$600B    XCOUNT   =$600C
   DELAY    =$600D    BTEMP     =$600E    MTEMP     =$600F    ELINE    =$6036
   ELINEA   =$6037    EDEPTH    =$6038    SUM       =$6039    COUNTER  =$603A
   DE       =$603B    PCOUNTER=$603C      PLINE     =$603D    PLINEA   =$603E
   PBYTE    =$603F    PDEPTH    =$6040    PSHPNO    =$6041    PTEMP    =$6042
   ST       =$6051    MSHPADR   =$6052    BSHPADR   =$6060    PSHPADR  =$606E
   PGM      =$607C    CLR1      =$6090    CLR       =$6094    LN       =$60B6
   PI       =$60CB    BI        =$60CE    PADDLE    =$60D1    PSTART   =$60D7
   PSTART1  =$60F4    PSTART2   =$60FC    BUL       =$610E    BULLET1  =$6129
   BULLET   =$612E    LONG      =$6153    MINITIAL=$6156      BINITIAL=$6165
   SINITIAL=$6170     PR        =$617B    PINITIAL=$6184      PCONT    =$6196
   PLOADSHP=$61AA     PLOADSHP1=$61BB     PDLE      =$61C6    PDLE1    =$61DC
   LOAD     =$61F6    MDRAW     =$6201    MDRAW1    =$6206    PDRAW    =$624B
   PDRAW1   =$6250    LOADBUL   =$6295    BDRAW     =$62B8    NOHIT    =$62D4
   COLLISION=$62DC    LG        =$62F5    BXDRAW    =$62F8    EXPLODE  =$6310
   SOUND    =$6353    SOUND1    =$6355    DRAWE1    =$6361    DRAWE2   =$6390
   INITE1   =$63CD    INITE2    =$63E1    INITE3    =$63F5    INITE4   =$6409
   SCORE    =$641D    C10       =$642E    STOP1     =$6449    STOP2    =$6462
   PRINT    =$646D    NSHAPE    =$649E    MSHAPE1   =$64EE    MSHAPE2  =$6515
   MSHAPE3  =$653C    MSHAPE4   =$6563    MSHAPE5   =$658A    MSHAPE6  =$65B1
   MSHAPE7  =$65D8    BSHAPE1   =$65FF    BSHAPE2   =$6600    BSHAPE3  =$6601
   BSHAPE4  =$6602    BSHAPE5   =$6603    BSHAPE6   =$6604    BSHAPE7  =$6605
   ESHAPE   =$6606    PSHAPE1   =$6630    PSHAPE2   =$663F    PSHAPE3  =$664E
   PSHAPE4  =$665D    PSHAPE5   =$666C    PSHAPE6   =$667B    PSHAPE7  =$668A
   BYTETBL  =$6699    OFFSET    =$679C    HI        =$689F    LO       =$695F
   SPEAKER  =$C030    GRAPHICS=$C050      MIXOFF    =$C052    PAGE1    =$C054
   HIRES    =$C057    BUTTON    =$C061    PREAD     =$FB1E    WAIT     =$FCA8
```

Well, that's it. Hooray and huzzah. Pop the cork, sound the horn, raise the flag, lean back, light a cigar, and get reacquainted with your loved ones. But don't rest on your laurels too long—there's more to come for all you masochists out there.

In the last chapter, I will make specific suggestions for game modifications using routines discussed in both Part One and Part Two. For now, to get you started and to see how easy it is (and also just for the heck of it), I've decided to present one such modification. The modification is simple—the plane is drawn with the DRAW-DRAW protocol instead of DRAW-ERASE. We can do this because the plane itself is not involved in collision detection, but rather the bullet. Here is the flowchart:


LOAD SHAPE ADDRESSES INTO SHPADR
DISPLAY AND CLEAR SCREEN
DRAW BOTTOM LINE
ZERO PCOUNTER AND ST
SINITIAL
MINITIAL
PINITIAL
INC PCOUNTER
PCOUNTER >100?
Yes
No
STOP GAME
KEY PRESSED?
Yes
No
BINITIAL
READ PADDLE
BULLET ON?
No
Yes
BHORIZ = MHORIZ
DRAW MAN



INC PSHPNO
PSHPNO >6?
Yes
No
NEXT SCREEN BYTE
#$FF IN PSHPNO
END OF SCREEN?
No
Yes
ERASE MAN & PLANE
PINITIAL
INC PCOUNTER
PCOUNTER >100?
No
Yes
DRAW PLANE
CLICK SPEAKER EVERY OTHER CYCLE
BULLET ON?
Yes
No
BUTTON PRESSED?
Yes
No
SET BULLET ON
DELAY
ERASE MAN
AND BTEMP WITH SCREEN BYTE
COLLISION?
Yes
No
ERASE PLANE
INC SCORE
DISPLAY SCORE
SCORE = 100?
No
Yes
DISPLAY 100, SET ST = 1
EXPLOSION SHAPES & SOUNDS
ST = 1?
Yes
No
ERASE MAN
DRAW BULLET
DELAY
ERASE BULLET & MAN
MOVE BLINE UP 8 LINES
< 5 FROM TOP?
No
Yes

As you can see, very few changes are required and all relate to the plane erase, which is done with a separate PXDRAW routine using the EOR instruction. In contrast to Program 10-1, where the plane is erased before every paddle access, here the plane is erased at only two places—at the end of the screen and after a collision. You might also notice that the plane sound is a bit higher in pitch with faster clicks—this is because DRAW-DRAW takes less time than DRAW-ERASE.

The change, simple as it is, has resulted in a somewhat better program because the plane moves with less flicker. We'll see in the last chapter how we can effect even other modifications to make the program more interesting.

```
]PROGRAM 10-2
:ASM
                **** THE GAME WITH PLANE DRAWN WITH DRAW-DRAW ****
                         ORG   $6000
6000: 4C 7C 60           JMP   PGM
                MLINE    DS    1
                MLINEA   DS    1
                BLINE    DS    1
                DEPTH    DS    1
                MHORIZ   DS    1
                BHORIZ   DS    1
                HORIZB   DS    1
                HORIZM   DS    1
                BULON    DS    1
                XCOUNT   DS    1
                DELAY    DS    1
                BTEMP    DS    1
                MTEMP    DS    39
                ELINE    DS    1
                ELINEA   DS    1
                EDEPTH   DS    1
                SUM      DS    1
                COUNTER  DS    1
                DE       DS    1
                PCOUNTER DS    1
                PLINE    DS    1
                PLINEA   DS    1
                PBYTE    DS    1
                PDEPTH   DS    1
                PSHPNO   DS    1
                PTEMP    DS    15
                ST       DS    1
                GRAPHICS =     $C050
                MIXOFF   =     $C052
                HIRES    =     $C057
                PAGE1    =     $C054
                HIGH     =     $1B
                LOW      =     $1A
                WAIT     =     $FCA8
                PREAD    =     $FB1E
                BUTTON   =     $C061        ;BUTTON 0
                SPEAKER  =     $C030
                *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                *CONTINUE FOR ALL 7 SHAPES
6052: 32        MSHPADR  DFB   #<MSHAPE1
6053: 65                 DFB   #>MSHAPE1
6054: 59                 DFB   #<MSHAPE2
6055: 65                 DFB   #>MSHAPE2
```

```
6056: 80            DFB  #<MSHAPE3
6057: 65            DFB  #>MSHAPE3
6058: A7            DFB  #<MSHAPE4
6059: 65            DFB  #>MSHAPE4
605A: CE            DFB  #<MSHAPE5
605B: 65            DFB  #>MSHAPE5
605C: F5            DFB  #<MSHAPE6
605D: 65            DFB  #>MSHAPE6
605E: 1C            DFB  #<MSHAPE7
605F: 66            DFB  #>MSHAPE7
6060: 43   BSHPADR  DFB  #<BSHAPE1
6061: 66            DFB  #>BSHAPE1
6062: 44            DFB  #<BSHAPE2
6063: 66            DFB  #>BSHAPE2
6064: 45            DFB  #<BSHAPE3
6065: 66            DFB  #>BSHAPE3
6066: 46            DFB  #<BSHAPE4
6067: 66            DFB  #>BSHAPE4
6068: 47            DFB  #<BSHAPE5
6069: 66            DFB  #>BSHAPE5
606A: 48            DFB  #<BSHAPE6
606B: 66            DFB  #>BSHAPE6
606C: 49            DFB  #<BSHAPE7
606D: 66            DFB  #>BSHAPE7
606E: 74   PSHPADR  DFB  #<PSHAPE1
606F: 66            DFB  #>PSHAPE1
6070: 83            DFB  #<PSHAPE2
6071: 66            DFB  #>PSHAPE2
6072: 92            DFB  #<PSHAPE3
6073: 66            DFB  #>PSHAPE3
6074: A1            DFB  #<PSHAPE4
6075: 66            DFB  #>PSHAPE4
6076: B0            DFB  #<PSHAPE5
6077: 66            DFB  #>PSHAPE5
6078: BF            DFB  #<PSHAPE6
6079: 66            DFB  #>PSHAPE6
607A: CE            DFB  #<PSHAPE7
607B: 66            DFB  #>PSHAPE7
607C: AD 50 C0 PGM  LDA  GRAPHICS   ;HIRES,P.1
607F: AD 52 C0      LDA  MIXOFF
6082: AD 57 C0      LDA  HIRES
6085: AD 54 C0      LDA  PAGE1
6088: A9 00         LDA  #$00       ;CLEAR SCREEN 1
608A: 85 1A         STA  LOW
608C: A9 20         LDA  #$20
608E: 85 1B         STA  HIGH
6090: A0 00   CLR1  LDY  #$00
6092: A9 00         LDA  #$00
6094: 91 1A   CLR   STA  (LOW),Y
6096: C8            INY
6097: D0 FB         BNE  CLR
6099: E6 1B         INC  HIGH
609B: A5 1B         LDA  HIGH
609D: C9 40         CMP  #$40
609F: 90 EF         BLT  CLR1
60A1: A9 50         LDA  #$50       ;LOAD DELAY
60A3: 8D 0D 60      STA  DELAY
60A6: A2 B7         LDX  #$B7       ;DRAW BOTTOM LINE
60A8: A0 00         LDY  #$00
60AA: BD E3 68      LDA  HI,X
60AD: 85 1B         STA  HIGH
```

```
60AF: BD A3 69            LDA  LO,X
60B2: 85 1A               STA  LOW
60B4: A9 7F               LDA  #$7F
60B6: 91 1A        LN     STA  (LOW),Y
60B8: C8                  INY
60B9: C0 27               CPY  #$27
60BB: 90 F9               BLT  LN
60BD: A9 00               LDA  #$00
60BF: 8D 3C 60            STA  PCOUNTER
60C2: 8D 51 60            STA  ST
                   ********** MAIN PROGRAM **********
60C5: 20 70 61            JSR  SINITIAL   ;INITIALIZATION
60C8: 20 56 61            JSR  MINITIAL
60CB: 20 84 61     PI     JSR  PINITIAL
60CE: 20 65 61     BI     JSR  BINITIAL
60D1: 20 C6 61     PADDLE JSR  PDLE       ;READ PADDLE
60D4: 20 01 62            JSR  MDRAW      ;DRAW MAN
60D7: EE 41 60     PSTART INC  PSHPNO     ;FIRST SHAPE NUMBER TO ZERO
60DA: AD 41 60            LDA  PSHPNO
60DD: C9 07               CMP  #$07       ;DRAWN ALL 7 SHAPES?
60DF: 90 21               BLT  PSTART2    ;IF NO, DRAW PLANE
60E1: EE 3F 60            INC  PBYTE      ;IF YES, NEXT SCREEN BYTE
60E4: AD 3F 60            LDA  PBYTE
60E7: C9 26               CMP  #$26       ;END OF SCREEN?
60E9: 90 0F               BLT  PSTART1    ;IF NO, RESET SHAPE NO. &
                   *                       CONTINUE DRAW
60EB: 20 01 62            JSR  MDRAW      ;IF YES, ERASE MAN AND
60EE: CE 3F 60            DEC  PBYTE
60F1: 20 8F 62            JSR  PXDRAW     ERASE PLANE AND
60F4: 20 84 61            JSR  PINITIAL   INITIALIIZE PLANE AND
60F7: 4C D1 60            JMP  PADDLE     GO BACK TO PADDLE READ
60FA: A9 FF        PSTART1 LDA #$FF
60FC: 8D 41 60            STA  PSHPNO
60FF: 4C D7 60            JMP  PSTART
6102: 20 AA 61     PSTART2 JSR PLOADSHP
6105: 20 4B 62            JSR  PDRAW      ;DRAW PLANE
6108: EE 3B 60            INC  DE         ;ACCESS SPEAKER EVERY OTHER CYCLE
610B: AD 3B 60            LDA  DE
                   **
610E: 4A                  LSR             ;C=0 IF DE IS EVEN
610F: 90 03               BCC  BUL        ;C=1 IF DE IS ODD
6111: 2C 30 C0            BIT  SPEAKER
6114: AD 0B 60     BUL    LDA  BULON
6117: C9 01               CMP  #$01       ;IS BULLET ON?
6119: F0 16               BEQ  BULLET     ;IF YES, CONTINUE BULLET DRAW
611B: AD 61 C0            LDA  BUTTON     ;IF NO, IS BUTTON PRESSED?
611E: 30 0C               BMI  BULLET1    ;IF YES, DRAW BULLET
6120: AD 0D 60            LDA  DELAY      ;IF NO,
6123: 20 A8 FC            JSR  WAIT          DELAY AND
6126: 20 01 62            JSR  MDRAW            ERASE MAN AND
6129: 4C D1 60            JMP  PADDLE        READ PADDLE AGAIN
612C: A9 01        BULLET1 LDA #$01       ;SET BULLET ON
612E: 8D 0B 60            STA  BULON
6131: 20 D9 62     BULLET JSR  LOADBUL    ;LOAD BULLET SHAPE INTO BTEMP
6134: 20 FC 62            JSR  BDRAW      ;DRAW BULLET & TEST FOR COLLISION
6137: AD 0D 60            LDA  DELAY
613A: 20 A8 FC            JSR  WAIT       ;DELAY
613D: 20 3C 63            JSR  BXDRAW     ;ERASE BULLET
6140: 20 01 62            JSR  MDRAW      ;ERASE MAN
6143: AD 05 60            LDA  BLINE
6146: 38                  SEC
```

```
6147: E9 08             SBC  #$08       ;MOVE BLINE UP 8 LINES
6149: 8D 05 60          STA  BLINE
614C: C9 05             CMP  #$05       ;LESS THAN 5 LINES FROM TOP?
614E: 90 03             BLT  LONG       ;IF YES, TAKE BRANCH
6150: 4C D1 60          JMP  PADDLE     ;IF NO, READ PADDLE AGAIN
6153: 4C CE 60  LONG    JMP  BI
                ********** SUBROUTINES **********
6156: A9 AA     MINITIAL LDA #$AA
6158: 8D 03 60          STA  MLINE
615B: 8D 04 60          STA  MLINEA
615E: 18                CLC
615F: 69 0D             ADC  #$0D
6161: 8D 06 60          STA  DEPTH
6164: 60                RTS
                ***********************
6165: A9 00     BINITIAL LDA #$00       ;BULON = 0 IF
6167: 8D 0B 60          STA  BULON          BULLET NOT ON SCREEN
616A: A9 A4             LDA  #$A4
616C: 8D 05 60          STA  BLINE
616F: 60                RTS
                *********************
6170: A9 00     SINITIAL LDA #$00       ;SCORE DISPLAYS THREE 0'S
6172: 8D 39 60          STA  SUM
6175: 8D 3A 60          STA  COUNTER
6178: AA                TAX
6179: A0 11             LDY  #$11
617B: 20 B1 64  PR      JSR  PRINT
617E: C8                INY
617F: C0 14             CPY  #$14
6181: 90 F8             BLT  PR
6183: 60                RTS
                *********************
6184: A9 FF     PINITIAL LDA #$FF       ;PSHPNO LOADED WITH #$FF SO FIRST
                *                       INC PSHPNO WILL LOAD PSHPNO
                *                       WITH ZERO
6186: 8D 41 60          STA  PSHPNO
6189: EE 3C 60          INC  PCOUNTER   ;PINITIAL AND PCOUNTER ACCESSED
                *                       ONLY ON COLLISION OR
                *                       END OF SCREEN
618C: AD 3C 60          LDA  PCOUNTER
618F: C9 65             CMP  #$65       ;PCOUNTER MORE THAN 100?
6191: 90 03             BLT  PCONT      ;IF NO, CONTINUE P INITIALIZATION
6193: 4C A6 64          JMP  STOP2      ;IF YES, STOP GAME
6196: A9 00     PCONT   LDA  #$00
6198: 8D 3F 60          STA  PBYTE
619B: A9 08             LDA  #$08
619D: 8D 3E 60          STA  PLINEA
61A0: 8D 3D 60          STA  PLINE
61A3: 18                CLC
61A4: 69 05             ADC  #$05
61A6: 8D 40 60          STA  PDEPTH
61A9: 60                RTS
                ***********************
61AA: AD 41 60  PLOADSHP LDA PSHPNO
61AD: 0A                ASL
61AE: AA                TAX
61AF: BD 6E 60          LDA  PSHPADR,X
61B2: 85 1A             STA  LOW
61B4: BD 6F 60          LDA  PSHPADR+1,X
61B7: 85 1B             STA  HIGH
61B9: A0 00             LDY  #$00
```

```
61BB: B1 1A        PLOADSHP1 LDA (LOW),Y
61BD: 99 42 60               STA  PTEMP,Y
61C0: C8                     INY
61C1: C0 0F                  CPY  #$0F
61C3: 90 F6                  BLT  PLOADSHP1
61C5: 60                     RTS
                   **************************
61C6: A2 00        PDLE      LDX  #$00
61C8: 20 1E FB               JSR  PREAD       ;READ PADDLE 0
61CB: 98                     TYA
61CC: 8D 07 60               STA  MHORIZ      ;0-255 IN MHORIZ
61CF: AD 0B 60               LDA  BULON
61D2: C9 01                  CMP  #$01        ;IS BULLET ON?
61D4: F0 06                  BEQ  PDLE1       ;IF YES, TAKE BRANCH
61D6: AD 07 60               LDA  MHORIZ      ;IF NO, SET BHORIZ EQUAL
61D9: 8D 08 60               STA  BHORIZ           TO MHORIZ
61DC: AC 07 60     PDLE1     LDY  MHORIZ
61DF: B9 DD 66               LDA  BYTETBL,Y   ;CONVERT 0-255 TO 0-36 (BYTE)
61E2: 8D 0A 60               STA  HORIZM      ;MAN BYTE POSITION
61E5: B9 E0 67               LDA  OFFSET,Y    ;GET SHAPE NUMBER
61E8: 0A                     ASL              ;LOAD SHAPE INTO MTEMP
61E9: AA                     TAX
61EA: BD 52 60               LDA  MSHPADR,X
61ED: 85 1A                  STA  LOW
61EF: BD 53 60               LDA  MSHPADR+1,X
61F2: 85 1B                  STA  HIGH
61F4: A0 00                  LDY  #$00
61F6: B1 1A        LOAD      LDA  (LOW),Y
61F8: 99 0F 60               STA  MTEMP,Y
61FB: C8                     INY
61FC: C0 27                  CPY  #$27
61FE: 90 F6                  BLT  LOAD
6200: 60                     RTS
                   ************************
6201: A9 00        MDRAW     LDA  #$00
6203: 8D 0C 60               STA  XCOUNT
6206: AE 03 60     MDRAW1    LDX  MLINE
6209: AC 0A 60               LDY  HORIZM
620C: BD E3 68               LDA  HI,X
620F: 85 1B                  STA  HIGH
6211: BD A3 69               LDA  LO,X
6214: 85 1A                  STA  LOW
6216: AE 0C 60               LDX  XCOUNT
6219: B1 1A                  LDA  (LOW),Y
621B: 5D 0F 60               EOR  MTEMP,X
621E: 91 1A                  STA  (LOW),Y
6220: C8                     INY
6221: B1 1A                  LDA  (LOW),Y
6223: 5D 10 60               EOR  MTEMP+1,X
6226: 91 1A                  STA  (LOW),Y
6228: C8                     INY
6229: B1 1A                  LDA  (LOW),Y
622B: 5D 11 60               EOR  MTEMP+2,X
622E: 91 1A                  STA  (LOW),Y
6230: EE 0C 60               INC  XCOUNT
6233: EE 0C 60               INC  XCOUNT
6236: EE 0C 60               INC  XCOUNT
6239: EE 03 60               INC  MLINE
623C: AD 03 60               LDA  MLINE
623F: CD 06 60               CMP  DEPTH
6242: 90 C2                  BLT  MDRAW1
```

```
6244: AD 04 60           LDA  MLINEA     ;RESET LINE
6247: 8D 03 60           STA  MLINE
624A: 60                 RTS
                  ***********************
624B: A9 00       PDRAW  LDA  #$00
624D: 8D 0C 60           STA  XCOUNT
6250: AC 3F 60    PDRAW1 LDY  PBYTE
6253: AE 3D 60           LDX  PLINE
6256: BD E3 68           LDA  HI,X
6259: 85 1B              STA  HIGH
625B: BD A3 69           LDA  LO,X
625E: 85 1A              STA  LOW
6260: AE 0C 60           LDX  XCOUNT
6263: BD 42 60           LDA  PTEMP,X
6266: 91 1A              STA  (LOW),Y
6268: C8                 INY
6269: BD 43 60           LDA  PTEMP+1,X
626C: 91 1A              STA  (LOW),Y
626E: C8                 INY
626F: BD 44 60           LDA  PTEMP+2,X
6272: 91 1A              STA  (LOW),Y
6274: EE 0C 60           INC  XCOUNT
6277: EE 0C 60           INC  XCOUNT
627A: EE 0C 60           INC  XCOUNT
627D: EE 3D 60           INC  PLINE
6280: AD 3D 60           LDA  PLINE
6283: CD 40 60           CMP  PDEPTH
6286: 90 C8              BLT  PDRAW1
6288: AD 3E 60           LDA  PLINEA     ;RESET LINE
628B: 8D 3D 60           STA  PLINE
628E: 60                 RTS
                  *************************
628F: A9 00       PXDRAW LDA  #$00
6291: 8D 0C 60           STA  XCOUNT
6294: AC 3F 60    PXDRAW1 LDY PBYTE
6297: AE 3D 60           LDX  PLINE
629A: BD E3 68           LDA  HI,X
629D: 85 1B              STA  HIGH
629F: BD A3 69           LDA  LO,X
62A2: 85 1A              STA  LOW
62A4: AE 0C 60           LDX  XCOUNT
62A7: B1 1A              LDA  (LOW),Y
62A9: 5D 42 60           EOR  PTEMP,X
62AC: 91 1A              STA  (LOW),Y
62AE: C8                 INY
62AF: B1 1A              LDA  (LOW),Y
62B1: 5D 43 60           EOR  PTEMP+1,X
62B4: 91 1A              STA  (LOW),Y
62B6: C8                 INY
62B7: B1 1A              LDA  (LOW),Y
62B9: 5D 44 60           EOR  PTEMP+2,X
62BC: 91 1A              STA  (LOW),Y
62BE: EE 0C 60           INC  XCOUNT
62C1: EE 0C 60           INC  XCOUNT
62C4: EE 0C 60           INC  XCOUNT
62C7: EE 3D 60           INC  PLINE
62CA: AD 3D 60           LDA  PLINE
62CD: CD 40 60           CMP  PDEPTH
62D0: 90 C2              BLT  PXDRAW1
62D2: AD 3E 60           LDA  PLINEA
62D5: 8D 3D 60           STA  PLINE
```

```
62D8: 60                RTS
                  ************************
62D9: AC 08 60    LOADBUL LDY  BHORIZ      ;CONVERTS 0-255 TO
62DC: B9 DD 66            LDA  BYTETBL,Y       SCREEN BYTE (0-36)
62DF: 18                  CLC              ;ADD 2 TO ALIGN BULLET
62E0: 69 02               ADC  #$02            WITH GUN
62E2: 8D 09 60            STA  HORIZB      ;BULLET BYTE POSITION
62E5: B9 E0 67            LDA  OFFSET,Y    ;GET BULLET SHAPE NUMBER
62E8: 0A                  ASL              ;LOAD BULLET SHAPE INTO BTEMP
62E9: AA                  TAX
62EA: BD 60 60            LDA  BSHPADR,X
62ED: 85 1A               STA  LOW
62EF: BD 61 60            LDA  BSHPADR+1,X
62F2: 85 1B               STA  HIGH
62F4: A0 00               LDY  #$00
62F6: B1 1A               LDA  (LOW),Y
62F8: 8D 0E 60            STA  BTEMP
62FB: 60                  RTS
                  **********************
62FC: AE 05 60    BDRAW   LDX  BLINE
62FF: AC 09 60            LDY  HORIZB
6302: BD E3 68            LDA  HI,X
6305: 85 1B               STA  HIGH
6307: BD A3 69            LDA  LO,X
630A: 85 1A               STA  LOW
630C: B1 1A               LDA  (LOW),Y
630E: 2D 0E 60            AND  BTEMP       ;RESULT IS 0 IF NO COLLISION
6311: C9 00               CMP  #$00
6313: F0 03               BEQ  NOHIT
6315: 4C 20 63            JMP  COLLISION
6318: B1 1A       NOHIT   LDA  (LOW),Y     ;DRAW BULLET
631A: 4D 0E 60            EOR  BTEMP
631D: 91 1A               STA  (LOW),Y
631F: 60                  RTS
                  ***********************
6320: 20 8F 62    COLLISION JSR PXDRAW     ;ERASE PLANE
6323: EE 39 60            INC  SUM         ;ADD 1 TO SCORE
6326: 20 61 64            JSR  SCORE       ;DISPLAY SCORE
6329: 20 54 63            JSR  EXPLODE     ;EXPLOSION DISPLAY AND SOUND
632C: AD 51 60            LDA  ST          ;IF COUNT=100,
632F: C9 01               CMP  #$01        THEN GO TO
6331: F0 06               BEQ  LG          STOP PROGRAM
6333: 20 01 62            JSR  MDRAW       ;ERASE MAN
6336: 4C CB 60            JMP  PI          ;INITIALIZE P, B, AND READ PADDLE
6339: 4C A6 64    LG      JMP  STOP2
                  ***********************
633C: AE 05 60    BXDRAW  LDX  BLINE       ;BDRAW WITHOUT COLLISION TEST
633F: AC 09 60            LDY  HORIZB
6342: BD E3 68            LDA  HI,X
6345: 85 1B               STA  HIGH
6347: BD A3 69            LDA  LO,X
634A: 85 1A               STA  LOW
634C: B1 1A               LDA  (LOW),Y
634E: 4D 0E 60            EOR  BTEMP
6351: 91 1A               STA  (LOW),Y
6353: 60                  RTS
                  **********************
6354: 20 11 64    EXPLODE JSR  INITE1
6357: 20 A5 63            JSR  DRAWE1      ;DRAW
635A: 20 97 63            JSR  SOUND       ;EXPLOSION SOUND
635D: 20 11 64            JSR  INITE1
```

```
6360: 20 A5 63          JSR  DRAWE1     ;ERASE
6363: 20 25 64          JSR  INITE2
6366: 20 A5 63          JSR  DRAWE1     ;DRAW
6369: A9 BB             LDA  #$BB
636B: 20 A8 FC          JSR  WAIT
636E: 20 25 64          JSR  INITE2
6371: 20 A5 63          JSR  DRAWE1     ;ERASE
6374: 20 39 64          JSR  INITE3
6377: 20 A5 63          JSR  DRAWE1     ;DRAW
637A: A9 BB             LDA  #$BB
637C: 20 A8 FC          JSR  WAIT
637F: 20 39 64          JSR  INITE3
6382: 20 A5 63          JSR  DRAWE1     ;ERASE
6385: 20 4D 64          JSR  INITE4
6388: 20 D4 63          JSR  DRAWE2     ;DRAW
638B: A9 FF             LDA  #$FF
638D: 20 A8 FC          JSR  WAIT
6390: 20 4D 64          JSR  INITE4
6393: 20 D4 63          JSR  DRAWE2     ;ERASE
6396: 60                RTS
                 ************************
6397: A0 02      SOUND   LDY  #$02       ;EXPLOSION SOUND
6399: 2C 30 C0   SOUND1  BIT  SPEAKER
639C: A9 60             LDA  #$60
639E: 20 A8 FC          JSR  WAIT
63A1: 88                DEY
63A2: D0 F5             BNE  SOUND1
63A4: 60                RTS
                 ***********************
63A5: AC 09 60   DRAWE1  LDY  HORIZB     ;ROUTINE FOR FIRST 3
63A8: AE 36 60          LDX  ELINE           EXPLOSION SHAPES
63AB: BD E3 68          LDA  HI,X
63AE: 85 1B             STA  HIGH
63B0: BD A3 69          LDA  LO,X
63B3: 85 1A             STA  LOW
63B5: AE 0C 60          LDX  XCOUNT
63B8: B1 1A             LDA  (LOW),Y
63BA: 5D 4A 66          EOR  ESHAPE,X
63BD: 91 1A             STA  (LOW),Y
63BF: EE 0C 60          INC  XCOUNT
63C2: EE 36 60          INC  ELINE
63C5: AD 36 60          LDA  ELINE
63C8: CD 38 60          CMP  EDEPTH
63CB: 90 D8             BLT  DRAWE1
63CD: AD 37 60          LDA  ELINEA
63D0: 8D 36 60          STA  ELINE
63D3: 60                RTS
                 **********************
63D4: AC 09 60   DRAWE2  LDY  HORIZB     ;ROUTINE FOR FOURTH
63D7: AE 36 60          LDX  ELINE           EXPLOSION SHAPE
63DA: BD E3 68          LDA  HI,X
63DD: 85 1B             STA  HIGH
63DF: BD A3 69          LDA  LO,X
63E2: 85 1A             STA  LOW
63E4: AE 0C 60          LDX  XCOUNT
63E7: B1 1A             LDA  (LOW),Y
63E9: 5D 4A 66          EOR  ESHAPE,X
63EC: 91 1A             STA  (LOW),Y
63EE: EE 0C 60          INC  XCOUNT
63F1: C8                INY
63F2: AE 0C 60          LDX  XCOUNT
```

```
63F5: B1 1A              LDA  (LOW),Y
63F7: 5D 4A 66           EOR  ESHAPE,X
63FA: 91 1A              STA  (LOW),Y
63FC: EE 0C 60           INC  XCOUNT
63FF: EE 36 60           INC  ELINE
6402: AD 36 60           LDA  ELINE
6405: CD 38 60           CMP  EDEPTH
6408: 90 CA              BLT  DRAWE2
640A: AD 37 60           LDA  ELINEA
640D: 8D 36 60           STA  ELINE
6410: 60                 RTS
                ************************
6411: A9 00     INITE1   LDA  #$00       ;INITIALIZE FIRST EXPLOSION
6413: 8D 0C 60           STA  XCOUNT
6416: A9 09              LDA  #$09
6418: 8D 37 60           STA  ELINEA
641B: 8D 36 60           STA  ELINE
641E: 18                 CLC
641F: 69 05              ADC  #$05
6421: 8D 38 60           STA  EDEPTH
6424: 60                 RTS
6425: A9 05     INITE2   LDA  #$05       ;INITIALIZE SECOND EXPLOSION
6427: 8D 0C 60           STA  XCOUNT
642A: A9 09              LDA  #$09
642C: 8D 37 60           STA  ELINEA
642F: 8D 36 60           STA  ELINE
6432: 18                 CLC
6433: 69 05              ADC  #$05
6435: 8D 38 60           STA  EDEPTH
6438: 60                 RTS
6439: A9 0A     INITE3   LDA  #$0A       ;INITIALIZE THIRD EXPLOSION
643B: 8D 0C 60           STA  XCOUNT
643E: A9 05              LDA  #$05
6440: 8D 37 60           STA  ELINEA
6443: 8D 36 60           STA  ELINE
6446: 18                 CLC
6447: 69 08              ADC  #$08
6449: 8D 38 60           STA  EDEPTH
644C: 60                 RTS
644D: A9 12     INITE4   LDA  #$12       ;INITIALIZE FOURTH EXPLOSION
644F: 8D 0C 60           STA  XCOUNT
6452: A9 01              LDA  #$01
6454: 8D 37 60           STA  ELINEA
6457: 8D 36 60           STA  ELINE
645A: 18                 CLC
645B: 69 0C              ADC  #$0C
645D: 8D 38 60           STA  EDEPTH
6460: 60                 RTS
                ************************
6461: AD 39 60  SCORE    LDA  SUM        ;GET SCORE (0-9)
6464: C9 0A              CMP  #$0A       ;GREATER THAN 9?
6466: B0 0A              BGE  C10        ;IF YES, BRANCH
6468: 0A                 ASL             ;IF NO, MULTIPLY BY 8
6469: 0A                 ASL
646A: 0A                 ASL
646B: AA                 TAX
646C: A0 13              LDY  #$13       ;BYTE POSITION FOR FIRST DIGIT
646E: 20 B1 64           JSR  PRINT      ;PRINT DIGIT
6471: 60                 RTS
6472: EE 3A 60  C10      INC  COUNTER    ;INC COUNTER (INITIALLY 0)
6475: AD 3A 60           LDA  COUNTER
```

```
6478: C9 0A           CMP  #$0A        ;MORE THAN 9?
647A: B0 11           BGE  STOP1       ;IF YES, PRINT 100 AND STOP GAME
647C: 0A              ASL              ;IF NO, MULTIPLY BY 8
647D: 0A              ASL
647E: 0A              ASL
647F: AA              TAX
6480: A0 12           LDY  #$12        ;BYTE POSITION OF MIDDLE DIGIT
6482: 20 B1 64        JSR  PRINT       ;PRINT DIGIT
6485: A9 00           LDA  #$00        ;ZERO SUM AND
6487: 8D 39 60        STA  SUM            RETURN TO PRINT 0
648A: 4C 61 64        JMP  SCORE             IN FIRST DIGIT POSITION
648D: A2 08     STOP1 LDX  #$08        ;PRINT 100 IN COUNTER
648F: A0 11           LDY  #$11
6491: 20 B1 64        JSR  PRINT
6494: A2 00           LDX  #$00
6496: A0 12           LDY  #$12
6498: 20 B1 64        JSR  PRINT
649B: A0 13           LDY  #$13
649D: 20 B1 64        JSR  PRINT
64A0: A9 01           LDA  #$01        ;SET ST TO INDICATE
64A2: 8D 51 60        STA  ST          COUNTER=100
64A5: 60              RTS
64A6: 2C 00 C0  STOP2 BIT  $C000       ;ANY KEY PRESSED?
64A9: 10 FB           BPL  STOP2       ;IF NO, BRANCH BACK & WAIT
                *                      FOR KEYSTROKE
64AB: 2C 10 C0        BIT  $C010       ;IF YES, CLEAR KEYBOARD STROBE
64AE: 4C 7C 60        JMP  PGM          AND START PROGRAM OVER
                *************************
64B1: BD E2 64  PRINT LDA  NSHAPE,X    ;RETRIEVE NUMBER SHAPE
64B4: 99 D0 23        STA  $23D0,Y     ;LINE #$B8 (184)
64B7: BD E3 64        LDA  NSHAPE+1,X
64BA: 99 D0 27        STA  $27D0,Y     ;LINE #$B9 (185)
64BD: BD E4 64        LDA  NSHAPE+2,X
64C0: 99 D0 2B        STA  $2BD0,Y     ;LINE #$BA (186)
64C3: BD E5 64        LDA  NSHAPE+3,X
64C6: 99 D0 2F        STA  $2FD0,Y     ;LINE #$BB (187)
64C9: BD E6 64        LDA  NSHAPE+4,X
64CC: 99 D0 33        STA  $33D0,Y     ;LINE #$BC (188)
64CF: BD E7 64        LDA  NSHAPE+5,X
64D2: 99 D0 37        STA  $37D0,Y     ;LINE #$BD (189)
64D5: BD E8 64        LDA  NSHAPE+6,X
64D8: 99 D0 3B        STA  $3BD0,Y     ;LINE #$BE (190)
64DB: BD E9 64        LDA  NSHAPE+7,X
64DE: 99 D0 3F        STA  $3FD0,Y     ;LINE #$BF (191)
64E1: 60              RTS
                ***************************
64E2: 00 1C 22  NSHAPE HEX 001C22222222221C    ;NUMBER SHAPES - "0"
64E5: 22 22 22 22 1C
64EA: 00 08 0C         HEX 00080C080808081C    ;"1"
64ED: 08 08 08 08 1C
64F2: 00 1C 22         HEX 001C22201008043E    ;"2"
64F5: 20 10 08 04 3E
64FA: 00 1C 22         HEX 001C22201C20221C    ;"3"
64FD: 20 1C 20 22 1C
6502: 00 10 18         HEX 00101814123E1010    ;"4"
6505: 14 12 3E 10 10
650A: 00 3E 02         HEX 003E021E2020201E    ;"5"
650D: 1E 20 20 20 1E
6512: 00 1C 22         HEX 001C22021E22221C    ;"6"
6515: 02 1E 22 22 1C
651A: 00 3E 20         HEX 003E201008040404    ;"7"
```

```
651D: 10 08 04 04 04
6522: 00 1C 22   589           HEX  001C22221C22221C      ;"8"
6525: 22 1C 22 22 1C
652A: 00 1C 22   590           HEX  001C22223C20221C      ;"9"
652D: 22 3C 20 22 1C
6532: 00 0E 01   591  MSHAPE1  HEX  000E01000E01000E01    ;MAN SHAPE TABLES
6535: 00 0E 01 00 0E 01
653B: 00 44 01   592           HEX  004401007F00601F00
653E: 00 7F 00 60 1F 00
6544: 30 1F 00   593           HEX  301F00181F00001F00
6547: 18 1F 00 00 1F 00
654D: 00 1F 00   594           HEX  001F00001B00403100
6550: 00 1B 00 40 31 00
6556: 60 60 00   595           HEX  606000
6559: 00 1C 02   596  MSHAPE2  HEX  001C02001C02001C02
655C: 00 1C 02 00 1C 02
6562: 00 08 03   597           HEX  000803007E01003E00
6565: 00 7E 01 00 3E 00
656B: 00 3F 00   598           HEX  003F00403F00003E00
656E: 40 3F 00 00 3E 00
6574: 00 3E 00   599           HEX  003E00003600003600
6577: 00 36 00 00 36 00
657D: 00 63 00   600           HEX  006300
6580: 00 38 04   601  MSHAPE3  HEX  003804003804003804
6583: 00 38 04 00 38 04
6589: 00 10 06   602           HEX  001006007C03007C00
658C: 00 7C 03 00 7C 00
6592: 00 7C 00   603           HEX  007C00007E00007C00
6595: 00 7E 00 00 7C 00
659B: 00 38 00   604           HEX  003800003800006C00
659E: 00 38 00 00 6C 00
65A4: 00 46 01   605           HEX  004601
65A7: 00 70 08   606  MSHAPE4  HEX  007008007008007008
65AA: 00 70 08 00 70 08
65B0: 00 20 0C   607           HEX  00200C007807007801
65B3: 00 78 07 00 78 01
65B9: 00 78 01   608           HEX  007801007801007801
65BC: 00 78 01 00 78 01
65C2: 00 70 00   609           HEX  007000007000007000
65C5: 00 70 00 00 70 00
65CB: 00 70 00   610           HEX  007000
65CE: 00 60 11   611  MSHAPE5  HEX  006011006011006011
65D1: 00 60 11 00 60 11
65D7: 00 40 18   612           HEX  00401800700F007003
65DA: 00 70 0F 00 70 03
65E0: 00 70 03   613           HEX  007003007803007003
65E3: 00 78 03 00 70 03
65E9: 00 60 01   614           HEX  006001006001003003
65EC: 00 60 01 00 30 03
65F2: 00 18 06   615           HEX  001806
65F5: 00 40 23   616  MSHAPE6  HEX  004023004023004023
65F8: 00 40 23 00 40 23
65FE: 00 00 31   617           HEX  00003100601F006007
6601: 00 60 1F 00 60 07
6607: 00 70 07   618           HEX  007007007807006007
660A: 00 78 07 00 60 07
6610: 00 60 07   619           HEX  006007006006006006
6613: 00 60 06 00 60 06
6619: 00 30 0C   620           HEX  00300C
661C: 00 00 47   621  MSHAPE7  HEX  000047000047000047
661F: 00 00 47 00 00 47
```

```
6625: 00 00 62                HEX  00006200403F00700F
6628: 00 40 3F 00 70 0F
662E: 00 58 0F                HEX  00580F004C0F00400F
6631: 00 4C 0F 00 40 0F
6637: 00 40 0F                HEX  00400F00400D006018
663A: 00 40 0D 00 60 18
6640: 00 30 30                HEX  003030
6643: 01            BSHAPE1   HEX  01              ;BULLET SHAPES
6644: 02            BSHAPE2   HEX  02
6645: 04            BSHAPE3   HEX  04
6646: 08            BSHAPE4   HEX  08
6647: 10            BSHAPE5   HEX  10
6648: 20            BSHAPE6   HEX  20
6649: 40            BSHAPE7   HEX  40
664A: 28 22 1A      ESHAPE    HEX  28221A2514      ;EXPLOSION SHAPES - NO. 1
664D: 25 14
664F: 2C 52 44                HEX  2C5244320C          ;NO. 2
6652: 32 0C
6654: 38 3E 7F                HEX  383E7F7E7E3F3F1C    ;NO. 3
6657: 7E 7E 3F 3F 1C
665C: 18 06 7C                HEX  18067C0F7C3F7E3F    ;NO. 4
665F: 0F 7C 3F 7E 3F
6664: 7C 7F 7C                HEX  7C7F7C3F7E3F7F1F
6667: 3F 7E 3F 7F 1F
666C: 7E 0F 7C                HEX  7E0F7C1F700F4003
666F: 1F 70 0F 40 03
6674: 02 00 00      PSHAPE1   HEX  0200000600007E1F00      ;PLANE SHAPES
6677: 06 00 00 7E 1F 00
667D: 7E 37 00                HEX  7E37007E7F00
6680: 7E 7F 00
6683: 04 00 00      PSHAPE2   HEX  0400000C00007C3F00
6686: 0C 00 00 7C 3F 00
668C: 7C 6F 00                HEX  7C6F007C7F01
668F: 7C 7F 01
6692: 08 00 00      PSHAPE3   HEX  080000180000787F00
6695: 18 00 00 78 7F 00
669B: 78 5F 01                HEX  785F01787F03
669E: 78 7F 03
66A1: 10 00 00      PSHAPE4   HEX  100000300000707F01
66A4: 30 00 00 70 7F 01
66AA: 70 3F 03                HEX  703F03707F07
66AD: 70 7F 07
66B0: 20 00 00      PSHAPE5   HEX  200000600000607F03
66B3: 60 00 00 60 7F 03
66B9: 60 7F 06                HEX  607F06607F0F
66BC: 60 7F 0F
66BF: 40 00 00      PSHAPE6   HEX  400000400100407F07
66C2: 40 01 00 40 7F 07
66C8: 40 7F 0D                HEX  407F0D407F1F
66CB: 40 7F 1F
66CE: 00 01 00      PSHAPE7   HEX  000100000300007F0F
66D1: 00 03 00 00 7F 0F
66D7: 00 7F 1B                HEX  007F1B007F3F
66DA: 00 7F 3F
```

BYTETBL
OFFSET
HI
LO

2659 bytes

```
Symbol table - numerical order:

   LOW      =$1A        HIGH     =$1B        MLINE    =$6003      MLINEA   =$6004
   BLINE    =$6005      DEPTH    =$6006      MHORIZ   =$6007      BHORIZ   =$6008
   HORIZB   =$6009      HORIZM   =$600A      BULON    =$600B      XCOUNT   =$600C
   DELAY    =$600D      BTEMP    =$600E      MTEMP    =$600F      ELINE    =$6036
   ELINEA   =$6037      EDEPTH   =$6038      SUM      =$6039      COUNTER  =$603A
   DE       =$603B      PCOUNTER=$603C       PLINE    =$603D      PLINEA   =$603E
   PBYTE    =$603F      PDEPTH   =$6040      PSHPNO   =$6041      PTEMP    =$6042
   ST       =$6051      MSHPADR  =$6052      BSHPADR  =$6060      PSHPADR  =$606E
   PGM      =$607C      CLR1     =$6090      CLR      =$6094      LN       =$60B6
   PI       =$60CB      BI       =$60CE      PADDLE   =$60D1      PSTART   =$60D7
   PSTART1  =$60FA      PSTART2  =$6102      BUL      =$6114      BULLET1  =$612C
   BULLET   =$6131      LONG     =$6153      MINITIAL=$6156       BINITIAL=$6165
   SINITIAL=$6170       PR       =$617B      PINITIAL=$6184       PCONT    =$6196
   PLOADSHP=$61AA       PLOADSHP1=$61BB       PDLE     =$61C6      PDLE1    =$61DC
   LOAD     =$61F6      MDRAW    =$6201      MDRAW1   =$6206      PDRAW    =$624B
   PDRAW1   =$6250      PXDRAW   =$628F      PXDRAW1  =$6294      LOADBUL  =$62D9
   BDRAW    =$62FC      NOHIT    =$6318      COLLISION=$6320      LG       =$6339
   BXDRAW   =$633C      EXPLODE  =$6354      SOUND    =$6397      SOUND1   =$6399
   DRAWE1   =$63A5      DRAWE2   =$63D4      INITE1   =$6411      INITE2   =$6425
   INITE3   =$6439      INITE4   =$644D      SCORE    =$6461      C10      =$6472
   STOP1    =$648D      STOP2    =$64A6      PRINT    =$64B1      NSHAPE   =$64E2
   MSHAPE1  =$6532      MSHAPE2  =$6559      MSHAPE3  =$6580      MSHAPE4  =$65A7
   MSHAPE5  =$65CE      MSHAPE6  =$65F5      MSHAPE7  =$661C      BSHAPE1  =$6643
   BSHAPE2  =$6644      BSHAPE3  =$6645      BSHAPE4  =$6646      BSHAPE5  =$6647
   BSHAPE6  =$6648      BSHAPE7  =$6649      ESHAPE   =$664A      PSHAPE1  =$6674
   PSHAPE2  =$6683      PSHAPE3  =$6692      PSHAPE4  =$66A1      PSHAPE5  =$66B0
   PSHAPE6  =$66BF      PSHAPE7  =$66CE      BYTETBL  =$66DD      OFFSET   =$67E0
   HI       =$68E3      LO       =$69A3      SPEAKER  =$C030      GRAPHICS=$C050
   MIXOFF   =$C052      PAGE1    =$C054      HIRES    =$C057      BUTTON   =$C061
   PREAD    =$FB1E      WAIT     =$FCA8
```

PART TWO

Advanced Techniques

11

Drawing in Color

A computer hack named Muller
Redesigned a dull program for color,
But his technique was so bad
The result was quite sad
For even in color it was duller.

Those of you who have your Apple hooked up to a color TV or monitor, consider yourself fortunate. There is hardly a game program, or any program for that matter, that uses hi-res graphics, that is not enhanced by a color display. In this chapter we'll look first at the mechanics of color production on the Apple and then see how to animate color shapes. We'll also discuss special problems that arise when testing color shapes for collisions. In the last chapter, I'll make some specific suggestions about using color to enhance the game program.

APPLE COLOR

Apple advertises that the hi-res screen can display six colors, but two of these are black and white. Pretty sneaky, eh? There are in fact only four colors available and they are blue, green, violet, and orange. This is not a particular drawback since, as you would see by examining commercial games, quite a lot can be done with just these few colors. For example, one of the most popular Apple games, Flight Simulator II, uses violet for water, blue for sky, green for ground, orange and blue for instruments, and violet for runway lights in night-time simulation. This works so well that one hardly notices only four colors are used, and this is the rule rather than the exception.

There are two principles involved in hi-res color production. One, if you don't have a color TV or monitor, you won't see color. This point is of such fundamental importance you should make sure you understand it before going any further. Got it? OK. The second principle is that a color shape is produced by plotting in alternate bit positions, that is, in every other column—bits next to each other produce white. In fact, white is produced only by adjacent bits—a single isolated bit is always in one of the four colors. The particular color produced depends on which columns are used, odd or even, and whether the high bit is set.

Let's discuss these points in some detail. First, the high or most significant bit, which is the left most bit in a byte, is the bit, you will remember, that is not plotted on the screen. Up till now, we've always set the high bit to zero for all our shapes. If the high bit is set to 1, the shape doesn't change, but the shape byte does. For example:

HIGH BIT (BIT 7) NOT SET

7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	1

SHAPE ON SCREEN

1	2	4	8	1	2	4	(8)
1	0	1	0	1	0	0	(0)

$\underbrace{1\ 0\ 1\ 0}\ \underbrace{1\ 0\ 0\ (0)}$ → #$15

HIGH BIT (BIT 7) SET

7	6	5	4	3	2	1	0
1	0	0	1	0	1	0	1

SHAPE ON SCREEN

1	2	4	8	1	2	4	(8)
1	0	1	0	1	0	0	(1)

$\underbrace{1\ 0\ 1\ 0}\ \underbrace{1\ 0\ 0\ (1)}$ → #$95

Thus, when the high bit is set, you use it to determine the hex value, remembering that the bit itself does not appear in the shape (actually, if you look carefully on a monochrome monitor, you'll see that dots plotted with a high bit set byte appear about one-half bit position over). This is why #$80 is equivalent to #$00 in terms of the shape produced, which in this case is no shape, i.e., black. Similarly, #$7F and #$FF will both produce the same white line. Apple refers to these colors as White 1 and White 2 and Black 1 and Black 2 (now we have eight hi-res colors, right?). Ordinarily, one uses black and white with the high bit off to eliminate any problems with detecting collisions with colors that have the high bit set.

As far as odd-even columns are concerned, we use the convention of numbering the first screen bit position at the left of the screen as 0 or the start of the even columns, and the second position as 1 or the start of the odd columns.

The four hi-res colors are produced by the following combinations:

Even columns—high bit not set—violet
high bit set—blue

Odd columns—high bit not set—green
high bit set—orange

Example

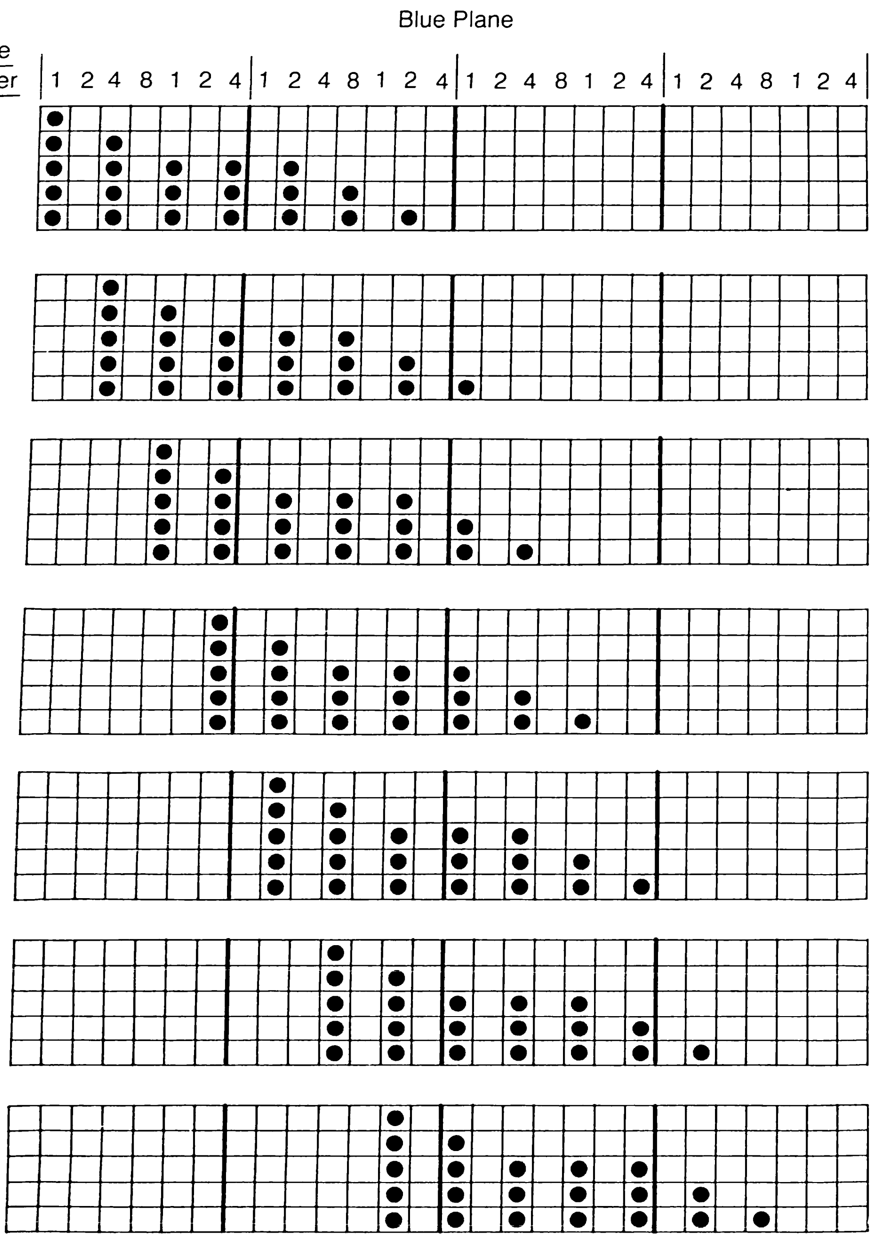
Shape on screen	*High bit not set*	*High bit set*
1 0 1 0 1 0 1	#$55 Violet	#$D5 Blue
0 1 0 1 0 1 0	#$2A Green	#$AA Orange

It should be emphasized that the odd-even column assignments always refer to the leftmost screen byte (byte 0). Thus if 1010101 is plotted in screen byte 1, the color will be green or orange, not violet or blue. That's all there is to it, but before we go on to the animation routines, there are two points we must mention. First, because we're plotting shapes as whole bytes, certain color combinations are not allowed. Any contiguous line cannot contain both violet and blue or green and orange because either the high bit is on or it isn't for the particular shape byte. Second, because we're plotting in alternate columns, the 280 dot horizontal resolution of the hi-res screen is reduced by half to 140 dots (the vertical resolution is not affected). This is not as bad as it seems because drawing shapes in different colors often produces an illusion of greater resolution than there really is because of the color contrasts. However, on a black and white monitor or TV, the loss of resolution is readily apparent as color shapes appear to be composed of dotted lines.

COLOR ANIMATION

The major problem in animating color shapes is maintaining the color throughout the screen range (you don't have to do this, but if you don't the result is mighty strange). This is not a problem for vertical animation because the shape bits always maintain their even or odd column assignments. The problem arises, as you might expect, only when dealing with movement that involves a horizontal vector; here, moving a shape in 1-bit moves would result in the bits occupying the wrong columns every other move. Fortunately, the solution to this problem is easy—we simply move the shape 2-bit positions at a time rather than 1; in this way the correct column assignments are always retained. Before we go on to discuss the details, it should be mentioned that 2-bit moves are also often useful for animating non-color shapes if we want, for example, to speed up the animation. The increase in jumpiness that results is generally acceptable. Therefore, the discussion that follows is applicable for both color and black and white animation.

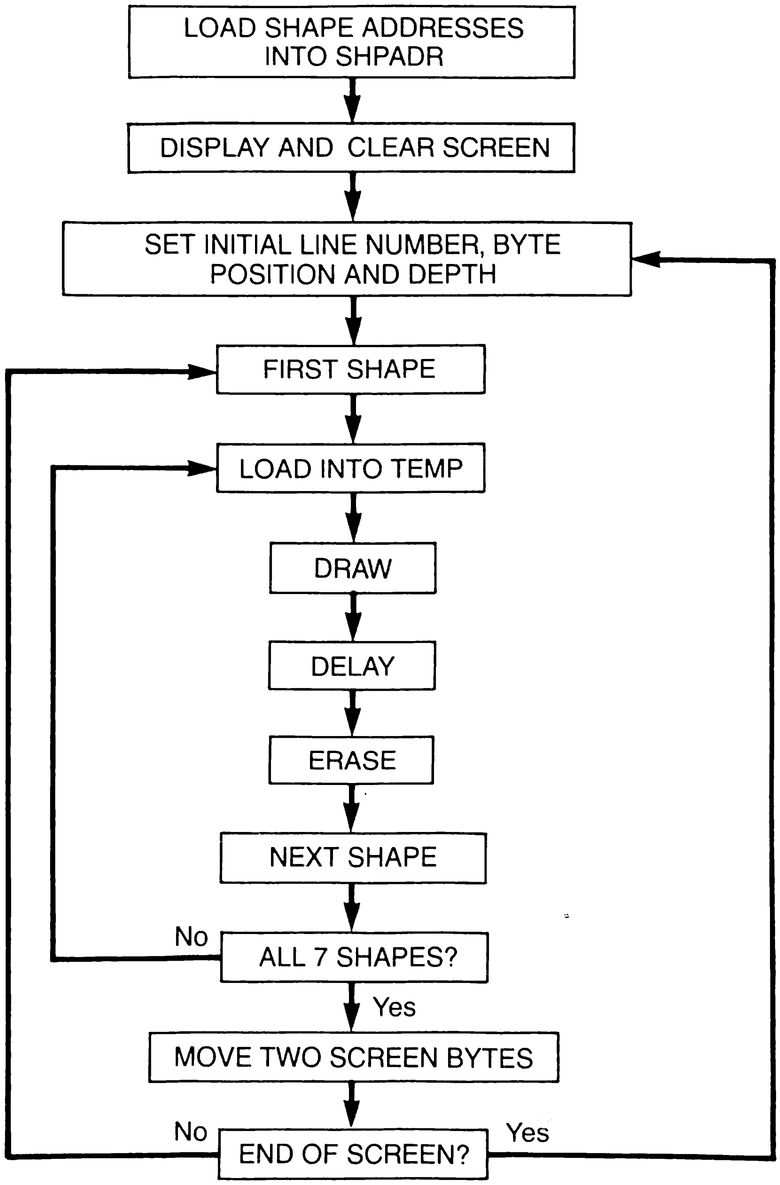
In the next program (Program 11-1) we're going to move a blue plane shape continuously across the screen at the same horizontal line position. The plane shape tables and shape bytes are as follows:



Shape Number	Shape Table			
0	81	80	80	80
	85	80	80	80
	D5	82	80	80
	D5	8A	80	80
	D5	AA	80	80
1	84	80	80	80
	94	80	80	80
	D4	8A	80	80
	D4	AA	80	80
	D4	AA	81	80
2	90	80	80	80
	D0	80	80	80
	D0	AA	80	80
	D0	AA	81	80
	D0	AA	85	80
3	C0	80	80	80
	C0	82	80	80
	C0	AA	81	80
	C0	AA	85	80
	C0	AA	95	80
4	80	82	80	80
	80	8A	80	80
	80	AA	85	80
	80	AA	95	80
	80	AA	D5	80
5	80	88	80	80
	80	A8	80	80
	80	A8	95	80
	80	A8	D5	80
	80	A8	D5	82
6	80	A0	80	80
	80	A0	81	80
	80	A0	D5	80
	80	A0	D5	82
	80	A0	D5	8A

There are several things that should be noted about these shape tables. Because we want the plane to be blue, the dots are plotted in the even columns only and the shape bytes represent the fact that the high bit is set. Also, note that although the shape itself is 2-screen-bytes wide, the shape table is 4-bytes wide to accommodate all seven shapes. Thus, a general principle—when moving a shape horizontally 2-bit positions at a time, two extra screen bytes in the direction of movement must be included in the shape tables instead of the one extra that we use for 1-bit moves. This necessitates a change in our usual drawing routine. In the MAIN PROGRAM of Program 11-1, when we've finished with all seven shapes, we increment the screen byte by 2 (lines 68 and 69) so that the next draw starts in the appropriate position. We can see this clearly in the shape diagrams above. If shape 0 is drawn in screen byte 0, the next shape 0 must be drawn in screen byte 2 and so on.

That's really all there is. The rest of the program needs no further explanation—we've seen it all before.




```
]PROGRAM 11-1
:ASM
                 *1 SHAPE HORIZONTAL*COLOR - BLUE
                 *2 BYTES WIDE, 5 LINES DEEP
                          ORG   $6000
6000: 4C 2C 60            JMP   PGM
                 LINE     DS    1
                 LINEA    DS    1
                 BYTE     DS    1
                 DEPTH    DS    1
                 XCOUNT   DS    1
                 SHPNO    DS    1
                 DELAY    DS    1
                 TEMP     DS    20
                 GRAPHICS =     $C050
                 MIXOFF   =     $C052
                 HIRES    =     $C057
                 PAGE1    =     $C054
                 HIGH     =     $1B
                 LOW      =     $1A
```

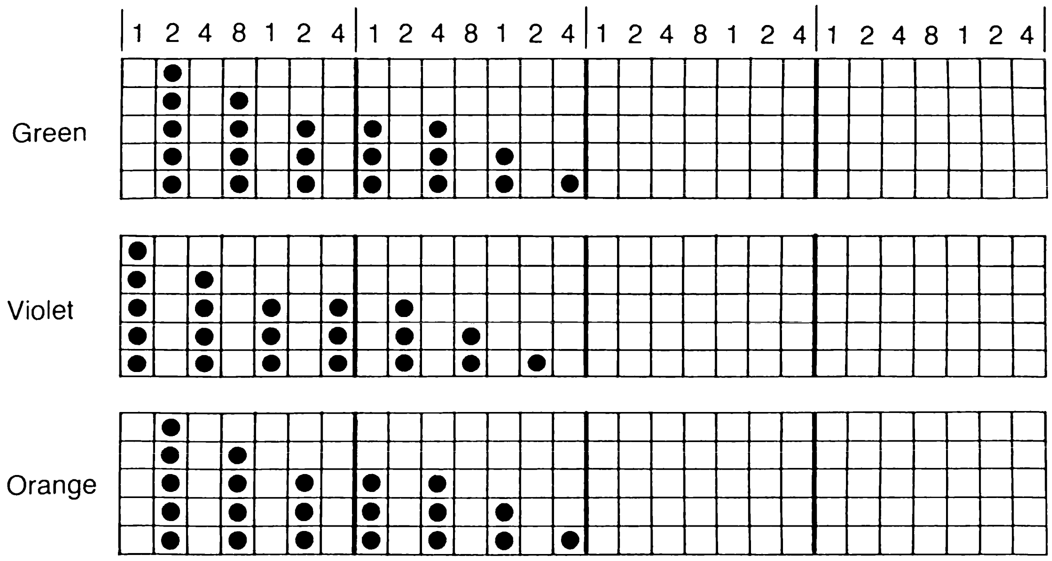
```
                     WAIT    =     $FCA8
                     *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                     *CONTINUE FOR ALL 7 SHAPES
601E: 0A             SHPADR  DFB   #<SHAPE1
601F: 61                     DFB   #>SHAPE1
6020: 1E                     DFB   #<SHAPE2
6021: 61                     DFB   #>SHAPE2
6022: 32                     DFB   #<SHAPE3
6023: 61                     DFB   #>SHAPE3
6024: 46                     DFB   #<SHAPE4
6025: 61                     DFB   #>SHAPE4
6026: 5A                     DFB   #<SHAPE5
6027: 61                     DFB   #>SHAPE5
6028: 6E                     DFB   #<SHAPE6
6029: 61                     DFB   #>SHAPE6
602A: 82                     DFB   #<SHAPE7
602B: 61                     DFB   #>SHAPE7
602C: AD 50 C0       PGM     LDA   GRAPHICS    ;HIRES,P.1
602F: AD 52 C0               LDA   MIXOFF
6032: AD 57 C0               LDA   HIRES
6035: AD 54 C0               LDA   PAGE1
6038: A9 00                  LDA   #$00        ;CLEAR SCREEN 1
603A: 85 1A                  STA   LOW
603C: A9 20                  LDA   #$20
603E: 85 1B                  STA   HIGH
6040: A0 00          CLR1    LDY   #$00
6042: A9 00                  LDA   #$00
6044: 91 1A          CLR     STA   (LOW),Y
6046: C8                     INY
6047: D0 FB                  BNE   CLR
6049: E6 1B                  INC   HIGH
604B: A5 1B                  LDA   HIGH
604D: C9 40                  CMP   #$40
604F: 90 EF                  BLT   CLR1
6051: A9 60                  LDA   #$60        ;LOAD DELAY
6053: 8D 09 60               STA   DELAY
                     ********** MAIN PROGRAM **********
6056: 20 87 60       START   JSR   INITIAL     ;SET INITIAL BYTE, LINE, DEPTH
6059: A9 00          START1  LDA   #$00        ;FIRST SHAPE NUMBER
605B: 8D 08 60               STA   SHPNO
605E: 20 99 60       START2  JSR   LOADSHP     ;LOAD SHAPE INTO TEMP
6061: 20 B5 60               JSR   DRAW        ;DRAW
6064: AD 09 60               LDA   DELAY       ;DELAY
6067: 20 A8 FC               JSR   WAIT
606A: 20 B5 60               JSR   DRAW        ;ERASE
606D: EE 08 60               INC   SHPNO       ;NEXT SHAPE NUMBER
6070: AD 08 60               LDA   SHPNO
6073: C9 07                  CMP   #$07        ;FINISHED ALL 7 SHAPES?
6075: 90 E7                  BLT   START2      ;IF NO, CONTINUE WITH NEXT SHAPE
6077: EE 05 60               INC   BYTE        ;IF YES, MOVE TWO BYTES
607A: EE 05 60               INC   BYTE
607D: AD 05 60               LDA   BYTE
6080: C9 26                  CMP   #$26        ;END OF SCREEN?
6082: 90 D5                  BLT   START1      ;IF NO, CONTINUE DRAW
6084: 4C 56 60               JMP   START       ;IF YES, START OVER
                     ********** SUBROUTINES **********
6087: A9 00          INITIAL LDA   #$00
6089: 8D 05 60               STA   BYTE
608C: 8D 03 60               STA   LINE
608F: 8D 04 60               STA   LINEA
6092: 18                     CLC
```

```
6093: 69 05                  ADC   #$05          ;DEPTH OF SHAPE
6095: 8D 06 60               STA   DEPTH
6098: 60                     RTS
                   *********************
6099: AD 08 60     LOADSHP   LDA   SHPNO         ;LOAD SHAPE INTO TEMP
609C: 0A                     ASL
609D: AA                     TAX
609E: BD 1E 60               LDA   SHPADR,X
60A1: 85 1A                  STA   LOW
60A3: BD 1F 60               LDA   SHPADR+1,X
60A6: 85 1B                  STA   HIGH
60A8: A0 00                  LDY   #$00
60AA: B1 1A        LOADSHP1  LDA   (LOW),Y
60AC: 99 0A 60               STA   TEMP,Y
60AF: C8                     INY
60B0: C0 14                  CPY   #$14
60B2: 90 F6                  BLT   LOADSHP1
60B4: 60                     RTS
                   ***********************
60B5: A9 00        DRAW      LDA   #$00
60B7: 8D 07 60               STA   XCOUNT
60BA: AC 05 60     DRAW1     LDY   BYTE
60BD: AE 03 60               LDX   LINE
60C0: BD 96 61               LDA   HI,X
60C3: 85 1B                  STA   HIGH
60C5: BD 56 62               LDA   LO,X
60C8: 85 1A                  STA   LOW
60CA: AE 07 60               LDX   XCOUNT
60CD: B1 1A                  LDA   (LOW),Y
60CF: 5D 0A 60               EOR   TEMP,X
60D2: 91 1A                  STA   (LOW),Y
60D4: C8                     INY
60D5: B1 1A                  LDA   (LOW),Y
60D7: 5D 0B 60               EOR   TEMP+1,X
60DA: 91 1A                  STA   (LOW),Y
60DC: C8                     INY
60DD: B1 1A                  LDA   (LOW),Y
60DF: 5D 0C 60               EOR   TEMP+2,X
60E2: 91 1A                  STA   (LOW),Y
60E4: C8                     INY
60E5: B1 1A                  LDA   (LOW),Y
60E7: 5D 0D 60               EOR   TEMP+3,X
60EA: 91 1A                  STA   (LOW),Y
60EC: EE 07 60               INC   XCOUNT
60EF: EE 07 60               INC   XCOUNT
60F2: EE 07 60               INC   XCOUNT
60F5: EE 07 60               INC   XCOUNT
60F8: EE 03 60               INC   LINE
60FB: AD 03 60               LDA   LINE
60FE: CD 06 60               CMP   DEPTH
6101: 90 B7                  BLT   DRAW1
6103: AD 04 60               LDA   LINEA
6106: 8D 03 60               STA   LINE          ;RESET LINE FOR NEXT CYCLE
6109: 60                     RTS
610A: 81 80 80     SHAPE1    HEX   8180808085808080D582 ;SHAPE TABLES
610D: 80 85 80 80 80 D5 82
6114: 80 80 D5               HEX   8080D58A8080D5AA8080
6117: 8A 80 80 D5 AA 80 80
611E: 84 80 80     SHAPE2    HEX   848080809480808 0D48A
6121: 80 94 80 80 80 D4 8A
6128: 80 80 D4               HEX   8080D4AA8080D4AA8180
```

```
612B: AA 80 80 D4 AA 81 80
6132: 90 80 80  138  SHAPE3    HEX  90808080D0808080D0AA
6135: 80 D0 80 80 80 D0 AA
613C: 80 80 D0  139            HEX  8080D0AA8180D0AA8580
613F: AA 81 80 D0 AA 85 80
6146: C0 80 80  140  SHAPE4    HEX  C0808080C0828080C0AA
6149: 80 C0 82 80 80 C0 AA
6150: 81 80 C0  141            HEX  8180C0AA8580C0AA9580
6153: AA 85 80 C0 AA 95 80
615A: 80 82 80  142  SHAPE5    HEX  80828080808A808080AA
615D: 80 80 8A 80 80 80 AA
6164: 85 80 80  143            HEX  858080AA958080AAD580
6167: AA 95 80 80 AA D5 80
616E: 80 88 80  144  SHAPE6    HEX  8088808080A8808080A8
6171: 80 80 A8 80 80 80 A8
6178: 95 80 80  145            HEX  958080A8D58080A8D582
617B: A8 D5 80 80 A8 D5 82
6182: 80 A0 80  146  SHAPE7    HEX  80A0808080A0818080A0
6185: 80 80 A0 81 80 80 A0
618C: D5 80 80  147            HEX  D58080A0D58280A0D58A
618F: A0 D5 82 80 A0 D5 8A
                     HI
                     LO

790 bytes

Symbol table - numerical order:

  LOW     =$1A      HIGH     =$1B     LINE     =$6003    LINEA    =$6004
  BYTE    =$6005    DEPTH    =$6006   XCOUNT   =$6007    SHPNO    =$6008
  DELAY   =$6009    TEMP     =$600A   SHPADR   =$601E    PGM      =$602C
  CLR1    =$6040    CLR      =$6044   START    =$6056    START1   =$6059
  START2  =$605E    INITIAL  =$6087   LOADSHP  =$6099    LOADSHP1=$60AA
  DRAW    =$60B5    DRAW1    =$60BA   SHAPE1   =$610A    SHAPE2   =$611E
  SHAPE3  =$6132    SHAPE4   =$6146   SHAPE5   =$615A    SHAPE6   =$616E
  SHAPE7  =$6182    HI       =$6196   LO       =$6256    GRAPHICS=$C050
  MIXOFF  =$C052    PAGE1    =$C054   HIRES    =$C057    WAIT     =$FCA8
```


Without changing anything in Program 11-1 except the shape bytes, we can draw the plane in any of the other three hi-res colors. Shown on the opposite page are the shapes and shape bytes for shape 0 for the plane in green, violet, and orange.

Shapes with multiple colors can be drawn quite easily, remembering though that a single byte can't contain two colors, one of which requires the high bit set and the other the high bit not set. This precludes a line in a shape within a single screen byte containing both violet and blue or green and orange. The line can, however, contain combinations of violet and green or blue and orange and, of course, different lines in the shape can contain any of the four colors. In addition, black and white can be placed anywhere. Note however that when combining colors, if two bits end up next to each other, white will be displayed in that region. The use of multiple colors, and the contrast they provide, goes a long way in mitigating the lower resolution of color displays.



Green	02	00	00	00
	0A	00	00	00
	2A	05	00	00
	2A	15	00	00
	2A	15	00	00
Violet	01	00	00	00
	05	00	00	00
	55	02	00	00
	55	0A	00	00
	55	2A	00	00
Orange	82	80	80	80
	8A	80	80	80
	AA	85	80	80
	AA	95	80	80
	AA	D5	80	80

Simple, yes? But of course for the privilege of drawing in color, there's a price to be paid, and I don't mean the cost of a color TV or monitor (actually, I don't know why this should be so but it seems to be a law of some kind—something about a free lunch?). In any case, we've already discussed one drawback, the lower resolution of color shapes. There is yet another that involves problems in collision detection and we'll get to that next.

COLLISION DETECTION WITH COLOR SHAPES

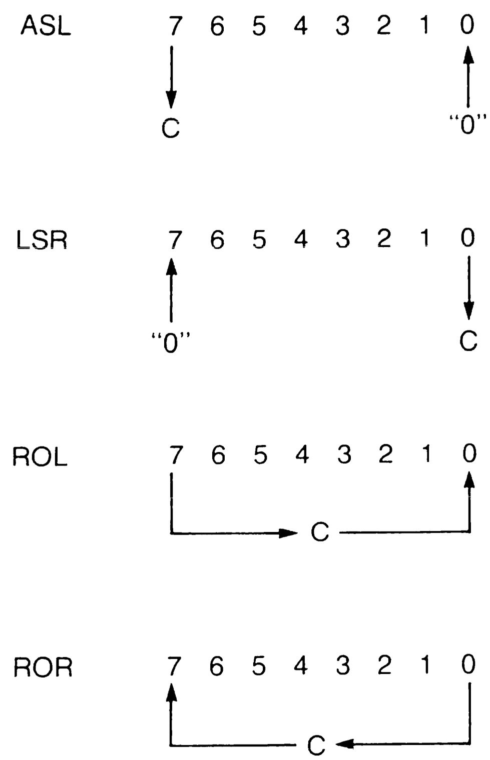
Collision detection with color shapes is difficult for two reasons; first, because such shapes contain "holes," and second, because of a problem relating to the high bit. Let's discuss the "hole" problem first.

Suppose we want to test for the collision of a vertically moving green shape with a violet one:

	7 6 5 4 3 2 1 0	*Shape on screen*
Shape 1—violet	0 1 0 1 0 1 0 1	1 0 1 0 1 0 1
AND with shape 2—green	0 0 1 0 1 0 1 0	0 1 0 1 0 1 0
Result	0 0 0 0 0 0 0 0	

The AND instruction returns a value of #$00 indicating no collision, but, of course, a collision should occur. The same situation holds for collisions between violet and orange shapes and between green and blue shapes, because they also occupy different columns. This is not a problem for violet and blue or green and orange shapes because here they occupy the same columns (on the other hand, blue and orange shapes will always indicate a collision, even when there shouldn't be one, because of the high bit problem we'll get to shortly). The same "hole" problem arises with horizontal movement, because color shapes are moved horizontally two bits at a time to maintain the alternate column assignments. We could get around the problem by changing colors, but this would limit our program options and also violate a basic creed of us assembly language programmers, to wit, "#$FF," or "Flexibility Forever," which translated means if we can overcome a limitation, let's do it.

When dealing with bits in the "wrong" set of alternate columns, the instructions that immediately come to mind are those that shift bits over one position; e.g., ASL (Arithmetic Shift Left), LSR (Logical Shift Right), ROR (ROtate Right), and ROL (ROtate Left).



When we do this kind of shifting, we have to make sure we can restore the original shape and color in preparation for the next shape draw and erase. This is done by storing the shape byte to be tested in the Accumulator, shifting the bits, and then storing the shifted byte into a memory location labeled, let's say, SHIFT (another clever nom de storage). Thus, the shape byte in the shape table is not affected by the shifting. The AND test is then done with the contents of SHIFT and the draw and erase with the shape byte from the shape table.

The instruction we're going to use is LSR because it's the only one that ensures the high bit will contain 0 after the shift—pushing a 1 into the high bit can present problems as we'll see below. Now suppose we want to collision-test a violet with a green shape as in the example in the beginning of this section. Let's see what happens if we LSR the violet shape before ANDing with the green shape:

	7 6 5 4 3 2 1 0
Shape 1—violet	0 1 0 1 0 1 0 1
LSR	0 0 1 0 1 0 1 0
AND with shape 2—green	0 0 1 0 1 0 1 0
Result—non-zero	0 0 1 0 1 0 1 0

Voila! We've detected a collision where there should be one. Let's see how this would look in a program (again, CMP #$00 is included to make the routine easier to read—it is not needed before a BEQ):

```
        LDA  SHAPE,X          ;GET SHAPE BYTE TO BE TESTED
        LSR                   ;SHIFT BITS RIGHT
        STA  SHIFT            ;STORE IN SHIFT
        LDA  (LOW),Y          ;GET SCREEN BYTE
        AND  SHIFT            ;AND WITH SHIFT CONTENTS
        CMP  #$00
        BEQ  NOHIT            ;JUMP TO NOHIT IF NO COLLISION
        JMP  COLLISION
NOHIT   Continue draw with SHAPE,X
```

Note that some assemblers require A in the operand column for LSR (and ASL, ROR, and ROL) when the bits in the Accumulator are to be shifted. The exact same procedure can be used for testing violet against orange and green against blue. When we get to blue vs. orange, however, we have a problem, because both colors have the high bit set and thus an AND test will always return a non-zero even when no collision is indicated. This occurs because both high bits are 1. Consider the following:

	7 6 5 4 3 2 1 0	*Shape on screen*
Shape 1—orange	1 0 0 0 1 0 1 0	0 1 0 1 0 0 0
AND with shape 2—blue	1 1 0 0 0 0 0 0	0 0 0 0 0 0 1
Result—non-zero	1 0 0 0 0 0 0 0	

Obviously a collision should not be detected, but it is because of the high bits. We might assume an LSR instruction would take care of this, because it places a zero in the high bit; but watch what happens:

	7 6 5 4 3 2 1 0
Shape 1—orange	1 0 0 0 1 0 1 0
LSR	0 1 0 0 0 1 0 1
AND with shape 2—blue	1 1 0 0 0 0 0 0
Result—non-zero	0 1 0 0 0 0 0 0

The high bit has been shifted into the shape byte and, in this case at least, an erroneous collision detection has occurred. What we want to do then is mask out the high bit before shifting. We do this by ANDing with #$7F. Thus:

	7 6 5 4 3 2 1 0
Shape 1—orange	1 0 0 0 1 0 1 0
AND with #$7F	0 1 1 1 1 1 1 1
Result—high bit 0	0 0 0 0 1 0 1 0
LSR	0 0 0 0 0 1 0 1
AND with shape 2—blue	1 1 0 0 0 0 0 0
Result—zero	0 0 0 0 0 0 0 0

In a program, the routine would look like this:

```
LDA SHAPE,X
AND #$7F
LSR
STA SHIFT
LDA (LOW),Y
AND SHIFT
CMP #$00
BEQ NOHIT
etc.
```


Even when testing blue or orange against a high-bit-not-set color, such as violet or green, it's still a good idea to mask out the high bit so that it doesn't get pushed into the shape byte. To summarize then:

Color to be tested (ANDed)	*Target color*	*Shift instructions*
violet	blue	none
green	orange	none
blue	violet	none
orange	green	none
green	violet or blue	LSR
violet	green or orange	LSR
blue	green or orange	AND #$7F, LSR
orange	violet or blue	AND #$7F, LSR
white (high bit off)	any color	none
any color	white (high bit off)	none

This is a fast and simple procedure, but not without its drawbacks. Because we're shifting the shape bits, in certain circumstances a collision detection will result when the shapes are not exactly at the collision site. For shapes moving vertically, this displacement will not exceed one bit position and this should certainly be tolerable in most circumstances. For shapes moving horizontally, the displacement can be as large as three bit positions, because the protocol is draw-erase-move two bit positions-LSR-AND test. We can reduce the displacement to one bit position by altering the protocol to the following; move two bit positions-draw-LSR-AND test-erase. But as mentioned, this displaced collision is not a problem in all cases—it depends on the shapes and which way they're moving.

A second method of collision detection with color shapes is one that seems to be favored by assembly language programmers, probably because the principle is simple and it works; there is also no problem with displaced collisions. The method involves setting up a second dummy shape table, identical to the first, except that the shape is drawn in white (high bit off), i.e., no "holes." The white shape is ANDed with the screen byte, and the color shape is used to draw and erase. For example, if the color shape is stored in COLOR and the white shape in WHITE, the routine would be as follows:

```
         LDA (LOW),Y
         AND WHITE,X
         CMP #$00
         BEQ NOHIT
         JMP COLLISION
NOHIT    Continue draw with COLOR,X
```

This method works for any color combination and eliminates the problem of the high bit. An obvious drawback, however, is that multiple shape tables have to be constructed for each shape involved in collision detection. This can eat up a lot of space for programs with many colliding shapes, not to mention the time involved in writing the program. There can also be a significant increase in execution time, depending on the type of animation involved. For vertical animation, this is not a significant problem—a single shape would have only two shape tables, one in color and the other in white, and the routine described above can be used without any modification. For horizontal movement, however, each shape would require 14 shape tables instead of 7, and if we use our usual TEMP loading routine, we would have to load another TEMP with the white shapes for each AND test. This could increase the execution time to intolerable levels and if so, we would then have to use routines that do not involve TEMP loading; that is, a separate draw routine for each shape as discussed in Chapter 5 (Program 5-2). This would further increase our program size, but then you can't have everything.

There is yet a third method we can use, which is both simple and fast, but has the limitation that the shape has to be all one color. What we do is use a single white shape table and then mask the shape to color. For example:

	7 6 5 4 3 2 1 0
White shape	0 0 0 1 1 1 0 0
AND with #$55	0 1 0 1 0 1 0 1
Result—violet shape (even columns)	0 0 0 1 0 1 0 0

A program using this routine would look like this:

```
         LDA (LOW),Y
         AND WHITE SHAPE
         CMP #$00
         BEQ NOHIT
         JMP COLLISION
NOHIT    LDA WHITE SHAPE
         AND #$55
         STA COLOR SHAPE
         LDA (LOW),Y
         EOR COLOR SHAPE
```

For a green shape, we would AND with #$2A, for orange with #$AA, and for blue with #$D5. This routine requires only one (white) shape table, but obviously can be used only with shapes of a single color, because each color requires a different value to be ANDed. The only exception would be if we are testing a vertically moving shape and expect no side collision. Here, only the top or bottom line (white shape) need be collision-tested and then, if no collision, the whole color shape can be drawn with a separate routine.

Which to use, LSR or white dummy shapes? That depends on the program and your own proclivities. Use whichever is easier and more appropriate. On the other hand, we could observe another assembly language credo, "#$EB," or "Easy is Better"—just change the shape colors to those that don't involve alternate columns.

Finally, let's discuss the game program for a moment. Suppose we draw the plane in blue. The bullet is a single dot and thus is either violet or green depending on the column in which it is drawn. Thus, half the time a collision will be missed, i.e., when the bullet is tested against a "hole" in the plane shape. Note that this is a special situation—ordinarily shapes are not just single dots. What to do? An LSR would be inappropriate—we would still miss collisions the other half of the time and if the shape byte were #$01, an LSR would empty the shape byte entirely. We could use a white dummy table—here the bullet would be 2-bits wide—but instead of going through all that trouble, why not just draw the bullet as a 2-bit wide white shape to begin with? Why not indeed. It works and looks fine—what more could we ask for?

12

Double Hi-Res Graphics and Animation

A computer artist from Lahore
Has only one problem; he can't draw.
Hi-res double
Gives him nothing but trouble,
Now he's twice as bad as before.

Those of you with Apple IIc's or extended memory Apple IIe's have probably sat up many a night wondering what to do with the extra memory these machines contain. You can't use it for your BASIC programs (it can be done but Apple won't tell you how) and only some commercial programs take advantage of it. But we're assembly language programmers and no part of Apple memory is inaccessible to us. In this chapter, we'll see how to use the extra memory to display and animate graphics in the double hi-res mode, both in color and black and white.

DOUBLE HI-RES—WHAT IT IS AND WHAT'S REQUIRED

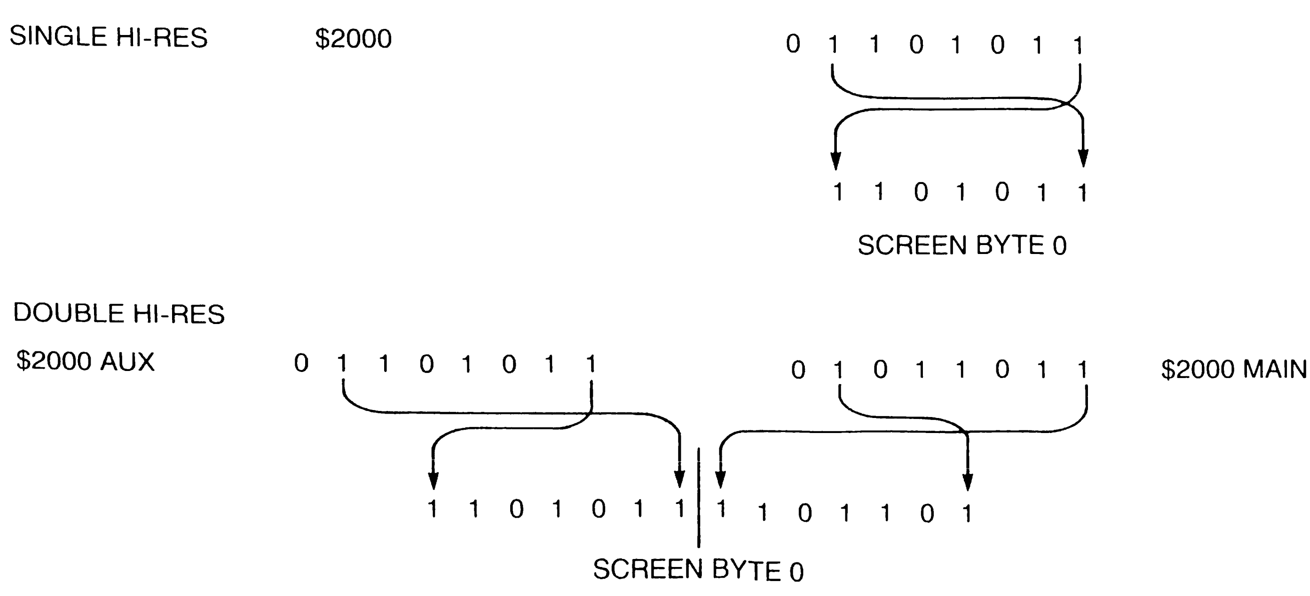
Not all Apples are capable of displaying double hi-res graphics—at the very least a minimum of 128K of memory is required. Apple IIc's come with 128K standard. Apple IIe's can be upgraded to 128K by adding an extended memory 80-column card (available from Apple and other companies) but double hi-res graphics also require that you have a revision B or later motherboard. You can tell what revision your motherboard is by checking the part number at the rear of the main circuit board. If the letter following the numbers is B, you're all set. If it is A, you have a revision A motherboard and double hi-res will not work. But don't despair—your Apple dealer will sell you a B motherboard at a price you can't refuse; all you have to do is learn how to program with one hand.

Another requirement, but one that is not absolutely essential, is a video monitor rather than a TV. You can use a black and white or color TV, but much detail will be lost, thus negating the increase in resolution and the details of color contrasts. You don't have to spend a lot of money for fancy monitors—I find the standard Apple monochrome monitor superb for double hi-res displays and even an inexpensive color monitor produces satisfactory results.

Double hi-res extends the horizontal resolution of the hi-res screen from 280 to 560 dots on a monochrome monitor—the vertical resolution remains the same at 192 lines. The 560 by 192 screen makes the Apple with double hi-res roughly equivalent to the hi-res mode on the IBM PC (600 by 200) and, as you might imagine, this increase in resolution can produce startlingly detailed graphics that make single hi-res appear rather crude. With color, the horizontal resolution is the same as single hi-res (140 dots), but with many more colors available and without the single hi-res color mixture limitations.

THE DOUBLE HI-RES SCREEN

The extra 64K of memory in 128K machines is essentially a mirror of the standard 64K memory block; that is, there are two of everything, including the hi-res screens. Let's label a hi-res screen from the standard memory as MAIN and the screen from the extra memory as AUX, for auxiliary. Each screen uses the same addresses; i.e., $2000 is the first screen byte position for hi-res Page 1 for both MAIN and AUX. For this reason, you have to specify which memory you're using before sending shape bytes to a hi-res screen location. Now remember that in single hi-res, 7 bits from a shape byte are plotted in a single screen byte. Thus, a shape byte sent to $2000 will be displayed in the first screen byte of hi-res Page 1. In double hi-res, each screen byte displays 7 bits from a shape byte from AUX and 7 bits from a shape byte from MAIN, the shape from AUX displayed in the first half of the screen byte. Thus, a shape byte sent to $2000 in AUX will be displayed in the first half of the first screen byte of Page 1 (byte 0) and a shape byte sent to $2000 in MAIN will be displayed in the second half (assuming, of course, that the double hi-res mode is selected).



Similarly, a shape byte sent to $2001 in AUX will be displayed in the first half of screen byte 1, and a shape byte sent to $2001 in MAIN will be displayed in the second half of screen byte 1, and so on. There are still just 40 screen bytes, but each can display up to 14 dots, which accounts for the 560 dot resolution ($14 \times 40 = 560$). All that needs to be done is to specify AUX or MAIN before sending the shape byte to the particular hi-res screen address—the double hi-res mode takes care of the plotting. Note that, as in single hi-res, the high bit does not appear in the shape. Not only that, but in double hi-res the high bit has nothing to do with color selection as we'll see in a later section.

THE DOUBLE HI-RES MODE

Strange as it may seem, you cannot draw in double hi-res unless you first set the double hi-res mode. The way to do this is buried deep within the Apple reference manual and if you're fond of frustration, you're welcome to try to dig it out, but why not just read on? The method, as you might suspect, involves accessing certain soft switches, some of which you've seen before.

Label	*Address*	*Access*	*Function*
GRAPHICS	$C050	LDA	Turns on graphics mode
MIXOFF	$C052	LDA	Selects full page graphics
HIRES	$C057	LDA	Selects hi-res mode
AN3	$C05E	LDA	Turns off annunciator 3
COL80	$C00D	STA	Selects 80 column mode
STORE80	$C001	STA	Changes functions of next switches
AUX	$C055	LDA	Selects AUX when STORE80 and HIRES on
MAIN	$C054	LDA	Selects MAIN

The routine for selecting double hi-res is as follows:

```
LDA GRAPHICS
LDA MIXOFF
LDA HIRES
STA STORE80
STA COL80
LDA AN3
```

Once this is done, LDA AUX selects the Page 1 hi-res screen from the auxiliary memory and LDA MAIN selects the same page from main memory. Thus, to clear both screens, we do:

```
LDA MAIN
JSR CLEAR
LDA AUX
JSR CLEAR
```

where CLEAR is our usual clear screen subroutine. It's simple when you know which switch to pull (push?), thus attesting to the old adage (which I just thought of) that computers imitate life.

DRAWING SHAPES

Drawing a shape on the double hi-res screen is relatively easy—all we have to do is determine in which half of the screen byte, AUX or MAIN, the shape bits are to be plotted and modify the draw routine accordingly. For example, let's look at how we would plot some simple shapes of varying lengths. To keep it really simple, we'll just plot some lines. For most of the programs in this chapter we'll be using single lines to illustrate the principles involved; however, the programs are designed in the usual way (i.e., XCOUNT, DEPTH, etc.) to allow the drawing of shapes with multiple lines, so our examples are applicable not just to lines, but to any shape.

Screen byte 12

AUX	MAIN
0 1 1 1 1 0 0	0 0 0 0 0 0 0

Here we're plotting shape byte #$1E just in the AUX portion of the screen byte. The draw routine would look like this (we're using the EOR method for illustration):

```
LDA AUX
LDA Screen byte
EOR #$1E
STA Screen byte
```


Suppose now we want to draw a line extending into the MAIN section:

Screen byte 12

AUX	MAIN
0 0 0 0 1 1 1	1 1 1 0 0 0 0

The draw routine would then be:

```
LDA AUX
LDA Screen byte
EOR #$70
STA Screen byte
LDA MAIN
LDA Screen byte     ;SAME SCREEN BYTE
EOR #$07
STA Screen byte
```

Now let's extend a line into the next screen byte:

Screen byte 12		*Screen byte 13*
AUX	MAIN	AUX
0 0 0 0 1 1 1	1 1 1 1 1 1 1	1 1 0 0 0 0 0

The draw routine would be:

```
LDA AUX
LDA Screen byte
EOR #$70
STA Screen byte
LDA MAIN
LDA Screen byte
EOR #$7F
STA Screen byte
INY                     ;NEXT SCREEN BYTE
LDA AUX
LDA Screen byte
EOR #$03
STA Screen byte
```

Note that we use INY to get to the next screen byte, because this is how we have always done it in our programs: Y is loaded with the value in BYTE and it is manipulated to access different screen bytes within the draw routine instead of BYTE itself to make erasing easier.

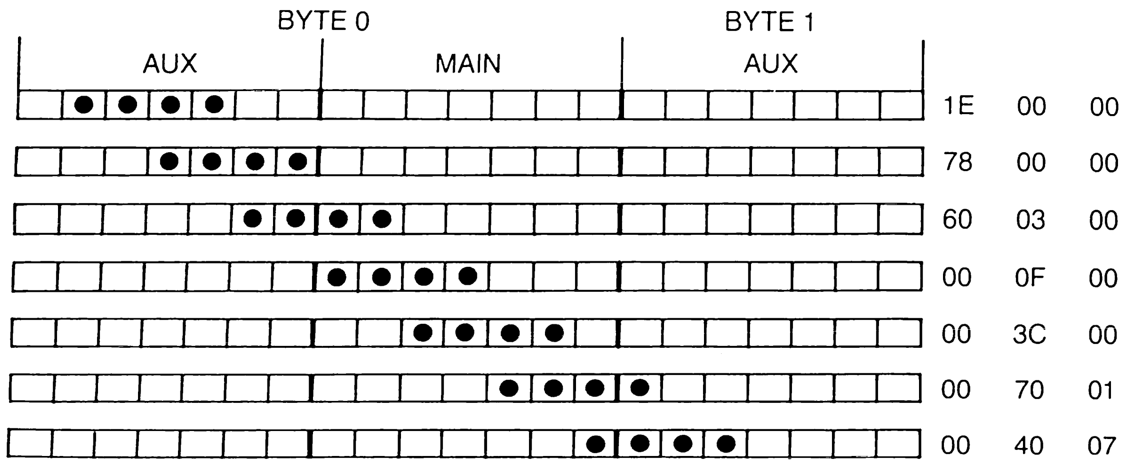
We can use a kind of shorthand to describe our double hi-res drawing routines. Thus, for the example above, A-M-INY-A. The same line starting in the MAIN section would use M-INY-A-M. A line extending over two whole screen bytes and starting in AUX would use A-M-INY-A-M, and so on. For a shape with multiple lines, we simply plot out the shape and design the draw routine based on the overall maximum shape width.

Now that we know how to display shapes on the double hi-res screen, let's see how to animate them.

ANIMATING SHAPES

Vertical animation, as usual, presents no problems. We just draw and erase the shape and change line positions; the shape bits always retain their column assignments. Non-vertical movement always contains a horizontal vector, and here things get more complicated, but not much more than with single hi-res horizontal movement. First of all, for greater simplicity, all our double hi-res horizontal animation will use 2-bit moves. One-bit moves are possible but involve greater complexity (14 shapes are required with different draw routines for each group of 7), and they are completely unnecessary because a 2-bit move in double hi-res is equivalent to a 1-bit move in single hi-res and this is certainly satisfactory.

With 2-bit moves, we need only 7 shapes. The technique is to examine the shape tables and devise the appropriate draw routine. Most everything else is the same as in our previous single hi-res programs. Let's consider the simplest example, a line occupying only half a screen byte and starting in the AUX section. Shown below are the shape tables for this line moving in 2-bit moves. (Note that as usual with 2-bit moves, 2 extra bytes have to be included in the shape tables in the direction of movement, but here the extra bytes are really half screen bytes, i.e., either a MAIN or AUX.)



Examination of the tables tells us the draw routine needed is A-M-INY-A. The tables also tell us that after seven shapes are drawn, we start over with the first shape in AUX in the next screen byte, and so we do an INC BYTE. This is in contrast to 2-bit moves in single hi-res where we have to move 2 bytes over after every seven shapes. Thus the protocol can be represented by A-M-INY-A—next screen byte—A-M-INY-A.

Let's now use these shapes in a program. The following program (Program 12-1) moves the line left to right across the screen in the same way that we moved the plane shape in previous programs. There is very little change from a single hi-res program, the major alteration being in the draw routine. The flowchart for Program 12-1 is on page 235.

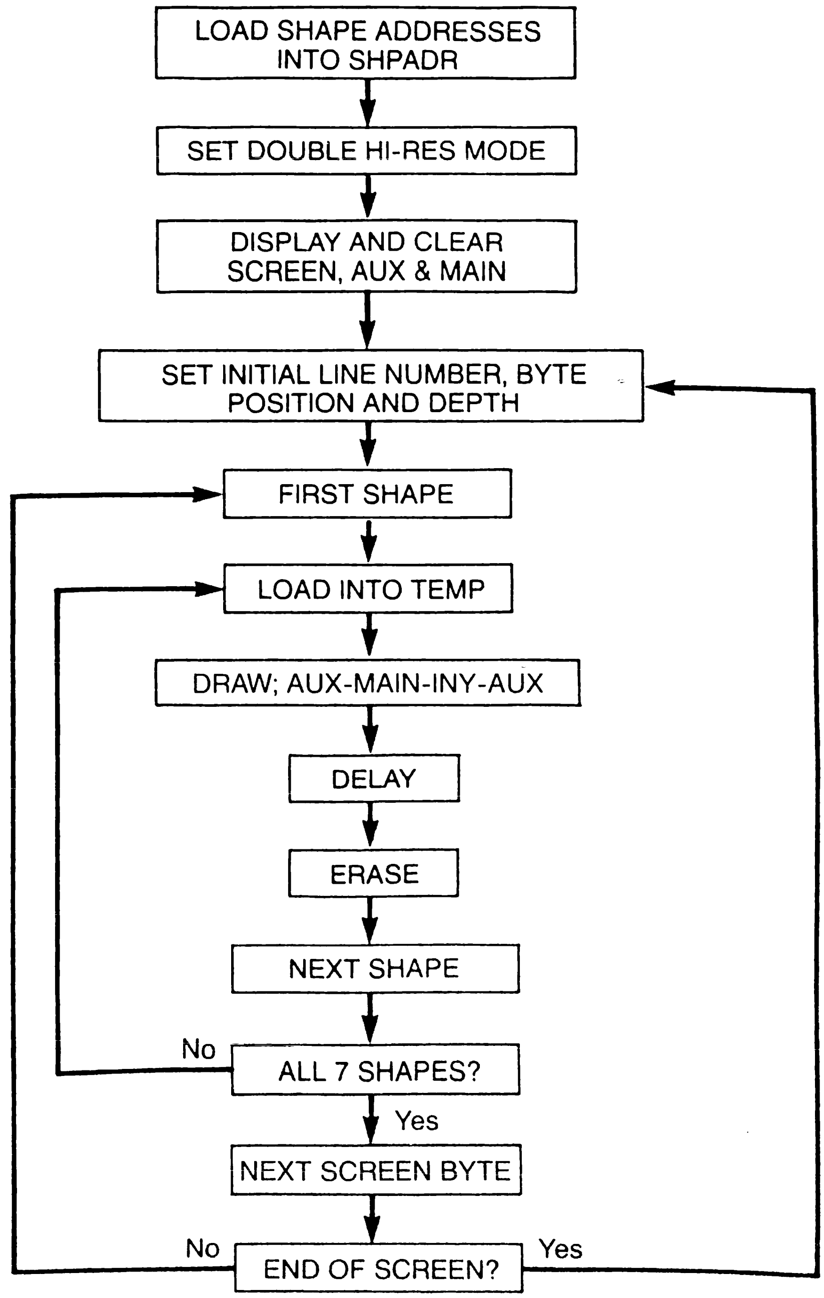

```
]PROGRAM 12-1
:ASM
                 ** DOUBLE HI-RES ** 2 BIT HORIZONTAL MOVE
                          ORG  $6000
6000: 4C 1B 60            JMP  PGM
                 LINE     DS   1
                 LINEA    DS   1
                 BYTE     DS   1
                 DEPTH    DS   1
                 XCOUNT   DS   1
                 SHPNO    DS   1
                 DELAY    DS   1
                 TEMP     DS   3
                 GRAPHICS =    $C050
                 MIXOFF   =    $C052
                 HIRES    =    $C057
                 AN3      =    $C05E
                 COL80    =    $C00D
                 STORE80  =    $C001
                 AUX      =    $C055
                 MAIN     =    $C054
                 HIGH     =    $1B
                 LOW      =    $1A
                 WAIT     =    $FCA8
                 *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                 *CONTINUE FOR ALL 7 SHAPES
600D: 09         SHPADR   DFB  #<SHAPE1
600E: 61                  DFB  #>SHAPE1
600F: 0C                  DFB  #<SHAPE2
6010: 61                  DFB  #>SHAPE2
```

```
6011: 0F              DFB  #<SHAPE3
6012: 61              DFB  #>SHAPE3
6013: 12              DFB  #<SHAPE4
6014: 61              DFB  #>SHAPE4
6015: 15              DFB  #<SHAPE5
6016: 61              DFB  #>SHAPE5
6017: 18              DFB  #<SHAPE6
6018: 61              DFB  #>SHAPE6
6019: 1B              DFB  #<SHAPE7
601A: 61              DFB  #>SHAPE7
601B: AD 50 C0   PGM     LDA  GRAPHICS
601E: AD 52 C0           LDA  MIXOFF
6021: AD 57 C0           LDA  HIRES
6024: 8D 01 C0           STA  STORE80
6027: 8D 0D C0           STA  COL80
602A: AD 5E C0           LDA  AN3
602D: AD 54 C0           LDA  MAIN
6030: 20 3C 60           JSR  CLEAR     ;CLEAR MAIN SCREEN
6033: AD 55 C0           LDA  AUX
6036: 20 3C 60           JSR  CLEAR     ;CLEAR AUX SCREEN
6039: 4C 56 60           JMP  DE
603C: A9 00      CLEAR   LDA  #00       ;CLEAR SCREEN SUBROUTINE
603E: 85 1A              STA  LOW
6040: A9 20              LDA  #$20
6042: 85 1B              STA  HIGH
6044: A0 00      CLR1    LDY  #00
6046: A9 00              LDA  #00
6048: 91 1A      CLR     STA  (LOW),Y
604A: C8                 INY
604B: D0 FB              BNE  CLR
604D: E6 1B              INC  HIGH
604F: A5 1B              LDA  HIGH
6051: C9 40              CMP  #$40
6053: 90 EF              BCC  CLR1
6055: 60                 RTS
6056: A9 60      DE      LDA  #$60      ;LOAD DELAY
6058: 8D 09 60           STA  DELAY
                 ********** MAIN PROGRAM **********
605B: 20 89 60   START   JSR  INITIAL   ;SET INITIAL BYTE, LINE, DEPTH
605E: A9 00      START1  LDA  #$00      ;FIRST SHAPE NUMBER
6060: 8D 08 60           STA  SHPNO
6063: 20 9B 60   START2  JSR  LOADSHP   ;LOAD SHAPE INTO TEMP
6066: 20 B7 60           JSR  DRAW      ;DRAW
6069: AD 09 60           LDA  DELAY     ;DELAY
606C: 20 A8 FC           JSR  WAIT
606F: 20 B7 60           JSR  DRAW      ;ERASE
6072: EE 08 60           INC  SHPNO     ;NEXT SHAPE NUMBER
6075: AD 08 60           LDA  SHPNO
6078: C9 07              CMP  #$07      ;FINISHED ALL 7 SHAPES?
607A: 90 E7              BLT  START2    ;IF NO, CONTINUE WITH NEXT SHAPE
607C: EE 05 60           INC  BYTE      ;IF YES, NEXT BYTE
607F: AD 05 60           LDA  BYTE
6082: C9 26              CMP  #$26      ;END OF SCREEN?
6084: 90 D8              BLT  START1    ;IF NO, CONTINUE DRAW
6086: 4C 5B 60           JMP  START     ;IF YES, START OVER
                 ********** SUBROUTINES **********
6089: A9 00      INITIAL LDA  #$00
608B: 8D 05 60           STA  BYTE
608E: 8D 03 60           STA  LINE
6091: 8D 04 60           STA  LINEA
6094: 18                 CLC
```

233

```
6095: 69 01                  ADC  #$01        ;DEPTH OF SHAPE
6097: 8D 06 60               STA  DEPTH
609A: 60                     RTS
              *********************
609B: AD 08 60    LOADSHP    LDA  SHPNO       ;LOAD SHAPE INTO TEMP
609E: 0A                     ASL
609F: AA                     TAX
60A0: BD 0D 60               LDA  SHPADR,X
60A3: 85 1A                  STA  LOW
60A5: BD 0E 60               LDA  SHPADR+1,X
60A8: 85 1B                  STA  HIGH
60AA: A0 00                  LDY  #$00
60AC: B1 1A       LOADSHP1   LDA  (LOW),Y
60AE: 99 0A 60               STA  TEMP,Y
60B1: C8                     INY
60B2: C0 03                  CPY  #$03
60B4: 90 F6                  BLT  LOADSHP1
60B6: 60                     RTS
              ** DRAW SUBROUTINE **
              ** AUX-MAIN-NEXT BYTE-AUX **
60B7: A9 00       DRAW       LDA  #$00
60B9: 8D 07 60               STA  XCOUNT
60BC: AC 05 60    DRAW1      LDY  BYTE
60BF: AE 03 60               LDX  LINE
60C2: BD 1E 61               LDA  HI,X
60C5: 85 1B                  STA  HIGH
60C7: BD DE 61               LDA  LO,X
60CA: 85 1A                  STA  LOW
60CC: AE 07 60               LDX  XCOUNT
60CF: AD 55 C0               LDA  AUX
60D2: B1 1A                  LDA  (LOW),Y
60D4: 5D 0A 60               EOR  TEMP,X
60D7: 91 1A                  STA  (LOW),Y
60D9: AD 54 C0               LDA  MAIN
60DC: B1 1A                  LDA  (LOW),Y
60DE: 5D 0B 60               EOR  TEMP+1,X
60E1: 91 1A                  STA  (LOW),Y
60E3: C8                     INY
60E4: AD 55 C0               LDA  AUX
60E7: B1 1A                  LDA  (LOW),Y
60E9: 5D 0C 60               EOR  TEMP+2,X
60EC: 91 1A                  STA  (LOW),Y
60EE: EE 07 60               INC  XCOUNT
60F1: EE 07 60               INC  XCOUNT
60F4: EE 07 60               INC  XCOUNT
60F7: EE 03 60               INC  LINE
60FA: AD 03 60               LDA  LINE
60FD: CD 06 60               CMP  DEPTH
6100: 90 BA                  BLT  DRAW1
6102: AD 04 60               LDA  LINEA
6105: 8D 03 60               STA  LINE        ;RESET LINE FOR NEXT CYCLE
6108: 60                     RTS
6109: 1E 00 00    SHAPE1     HEX  1E0000
610C: 78 00 00    SHAPE2     HEX  780000
610F: 60 03 00    SHAPE3     HEX  600300
6112: 00 0F 00    SHAPE4     HEX  000F00
6115: 00 3C 00    SHAPE5     HEX  003C00
6118: 00 70 01    SHAPE6     HEX  007001
611B: 00 40 07    SHAPE7     HEX  004007
                  HI
                  LO
```

```
670 bytes

Symbol table - numerical order:

   LOW      =$1A       HIGH     =$1B       LINE     =$6003     LINEA    =$6004
   BYTE     =$6005     DEPTH    =$6006     XCOUNT   =$6007     SHPNO    =$6008
   DELAY    =$6009     TEMP     =$600A     SHPADR   =$600D     PGM      =$601B
   CLEAR    =$603C     CLR1     =$6044     CLR      =$6048     DE       =$6056
   START    =$605B     START1   =$605E     START2   =$6063     INITIAL  =$6089
   LOADSHP  =$609B     LOADSHP1=$60AC      DRAW     =$60B7     DRAW1    =$60BC
   SHAPE1   =$6109     SHAPE2   =$610C     SHAPE3   =$610F     SHAPE4   =$6112
   SHAPE5   =$6115     SHAPE6   =$6118     SHAPE7   =$611B     HI       =$611E
   LO       =$61DE     STORE80 =$C001      COL80    =$C00D     GRAPHICS=$C050
   MIXOFF   =$C052     MAIN     =$C054     AUX      =$C055     HIRES    =$C057
   AN3      =$C05E     WAIT     =$FCA8
```



The program can be modified easily to move lines of any length by altering the draw routine. For a line occupying both AUX and MAIN of one screen byte, the protocol is A-M-INY-A-M—next screen byte—A-M-INY-A-M. For a line occupying AUX and MAIN of one screen byte and AUX of the next, we would use A-M-INY-A-M-INY-A—next screen byte—A-M-INY-A-M-INY-A, and so on.

To demonstrate how to draw complicated shapes, I've included the following program, which moves a spaceship vertically. Running this program will illustrate how neat double hi-res is compared to single hi-res.

```
]PROGRAM 12-2
:ASM
                      ** DOUBLE HI-RES * VERTICAL SPACESHIP
                                ORG  $6000
6000: 4C 09 60                  JMP  PGM
                      LINE      DS   1
                      LINEA     DS   1
                      BYTE      DS   1
                      DEPTH     DS   1
                      XCOUNT    DS   1
                      DELAY     DS   1
                      GRAPHICS  =    $C050
                      MIXOFF    =    $C052
                      HIRES     =    $C057
                      AN3       =    $C05E
                      COL80     =    $C00D
                      STORE80   =    $C001
                      AUX       =    $C055
                      MAIN      =    $C054
                      HIGH      =    $1B
                      LOW       =    $1A
                      WAIT      =    $FCA8
6009: AD 50 C0        PGM       LDA  GRAPHICS
600C: AD 52 C0                  LDA  MIXOFF
600F: AD 57 C0                  LDA  HIRES
6012: 8D 01 C0                  STA  STORE80
6015: 8D 0D C0                  STA  COL80
6018: AD 5E C0                  LDA  AN3
601B: AD 54 C0                  LDA  MAIN
601E: 20 2A 60                  JSR  CLEAR      ;CLEAR MAIN SCREEN
6021: AD 55 C0                  LDA  AUX
6024: 20 2A 60                  JSR  CLEAR      ;CLEAR AUX SCREEN
6027: 4C 44 60                  JMP  DE
602A: A9 00           CLEAR     LDA  #00        ;CLEAR SCREEN SUBROUTINE
602C: 85 1A                     STA  LOW
602E: A9 20                     LDA  #$20
6030: 85 1B                     STA  HIGH
6032: A0 00           CLR1      LDY  #00
6034: A9 00                     LDA  #00
6036: 91 1A           CLR       STA  (LOW),Y
6038: C8                        INY
6039: D0 FB                     BNE  CLR
603B: E6 1B                     INC  HIGH
603D: A5 1B                     LDA  HIGH
603F: C9 40                     CMP  #$40
6041: 90 EF                     BCC  CLR1
6043: 60                        RTS
6044: A9 60           DE        LDA  #$60       ;LOAD DELAY
6046: 8D 08 60                  STA  DELAY
```

```
                 ********** MAIN PROGRAM **********
6049: 20 6B 60   START    JSR  INITIAL    ;SET INITIAL BYTE, LINE, DEPTH
604C: 20 7D 60   START1   JSR  DRAW
604F: AD 08 60            LDA  DELAY
6052: 20 A8 FC            JSR  WAIT
6055: 20 7D 60            JSR  DRAW
6058: EE 06 60            INC  DEPTH
605B: EE 04 60            INC  LINEA
605E: AD 04 60            LDA  LINEA
6061: 8D 03 60            STA  LINE
6064: C9 B8               CMP  #$B8
6066: B0 E1               BGE  START
6068: 4C 4C 60            JMP  START1
                 ********** SUBROUTINES **********
606B: A9 00      INITIAL  LDA  #$00
606D: 8D 05 60            STA  BYTE
6070: 8D 03 60            STA  LINE
6073: 8D 04 60            STA  LINEA
6076: 18                  CLC
6077: 69 0D               ADC  #$0D       ;DEPTH OF SHAPE
6079: 8D 06 60            STA  DEPTH
607C: 60                  RTS
                 **********************
                 ** DRAW SUBROUTINE **
607D: A9 00      DRAW     LDA  #$00
607F: 8D 07 60            STA  XCOUNT
6082: AC 05 60   DRAW1    LDY  BYTE
6085: AE 03 60            LDX  LINE
6088: BD 25 61            LDA  HI,X
608B: 85 1B               STA  HIGH
608D: BD E5 61            LDA  LO,X
6090: 85 1A               STA  LOW
6092: AE 07 60            LDX  XCOUNT
6095: AD 55 C0            LDA  AUX
6098: B1 1A               LDA  (LOW),Y
609A: 5D E4 60            EOR  SHAPE,X
609D: 91 1A               STA  (LOW),Y
609F: AD 54 C0            LDA  MAIN
60A2: B1 1A               LDA  (LOW),Y
60A4: 5D E5 60            EOR  SHAPE+1,X
60A7: 91 1A               STA  (LOW),Y
60A9: C8                  INY
60AA: AD 55 C0            LDA  AUX
60AD: B1 1A               LDA  (LOW),Y
60AF: 5D E6 60            EOR  SHAPE+2,X
60B2: 91 1A               STA  (LOW),Y
60B4: AD 54 C0            LDA  MAIN
60B7: B1 1A               LDA  (LOW),Y
60B9: 5D E7 60            EOR  SHAPE+3,X
60BC: 91 1A               STA  (LOW),Y
60BE: C8                  INY
60BF: AD 55 C0            LDA  AUX
60C2: B1 1A               LDA  (LOW),Y
60C4: 5D E8 60            EOR  SHAPE+4,X
60C7: 91 1A               STA  (LOW),Y
60C9: AD 07 60            LDA  XCOUNT
60CC: 18                  CLC
60CD: 69 05               ADC  #$05
60CF: 8D 07 60            STA  XCOUNT
60D2: EE 03 60            INC  LINE
60D5: AD 03 60            LDA  LINE
```

```
60D8: CD 06 60  109         CMP  DEPTH
60DB: 90 A5     110         BLT  DRAW1
60DD: AD 04 60  111         LDA  LINEA
60E0: 8D 03 60  112         STA  LINE       ;RESET LINE FOR NEXT CYCLE
60E3: 60        113         RTS
60E4: 00 60 3F  114  SHAPE  HEX  00603F0000
60E7: 00 00
60E9: 00 50 56  115         HEX  0050560000
60EC: 00 00
60EE: 00 78 7F  116         HEX  00787F0100
60F1: 01 00
60F3: 01 07 00  117         HEX  0107000E08
60F6: 0E 08
60F8: 7E 40 3B  118         HEX  7E403B7007
60FB: 70 07
60FD: 7E 5C 52  119         HEX  7E5C527307
6100: 73 07
6102: 41 54 49  120         HEX  4154491208
6105: 12 08
6107: 40 5C 48  121         HEX  405C481300
610A: 13 00
610C: 40 00 00  122         HEX  4000001000
610F: 10 00
6111: 40 7F 7F  123         HEX  407F7F1F00
6114: 1F 00
6116: 20 0C 00  124         HEX  200C002300
6119: 23 00
611B: 10 0C 00  125         HEX  100C004300
611E: 43 00
6120: 1C 3F 60  126         HEX  1C3F604F03
6123: 4F 03
                     HI
                     LO

677 bytes

Symbol table - numerical order:

   LOW     =$1A      HIGH    =$1B      LINE    =$6003    LINEA   =$6004
   BYTE    =$6005    DEPTH   =$6006    XCOUNT  =$6007    DELAY   =$6008
   PGM     =$6009    CLEAR   =$602A    CLR1    =$6032    CLR     =$6036
   DE      =$6044    START   =$6049    START1  =$604C    INITIAL =$606B
   DRAW    =$607D    DRAW1   =$6082    SHAPE   =$60E4    HI      =$6125
   LO      =$61E5    STORE80 =$C001    COL80   =$C00D    GRAPHICS=$C050
   MIXOFF  =$C052    MAIN    =$C054    AUX     =$C055    HIRES   =$C057
   AN3     =$C05E    WAIT    =$FCA8
```


We've seen that drawing and animating shapes in double hi-res is relatively simple—just by examining the shape tables we can tell what kind of draw routine is required. The only difficulty is that the extra resolution afforded puts greater demands on our artistic talents, whatever they may be. But with this greater demand comes a greater opportunity and the extra work required is well worth the results.

DOUBLE HI-RES COLOR SHAPES

Displaying color in double hi-res requires quite a different procedure from single hi-res. First, the high bit has nothing to do with color selection—it is simply ignored. Second, colors are not produced by plotting dots in alternate rows, but rather are determined by the particular combination of 4 dots or bits displayed at particular positions on the screen. For example, if we were to place 0 0 0 1 in AUX1, the first screen position at the left of the screen, magenta would be displayed. If instead we plotted 0 1 0 0, we would get a dark green, 0 1 1 1 a yellow, and so on. As there are 16 combinations of 4 bits, 16 colors are available. However, one is white, one is black, and there are two greys, and so actually we have 13 colors to choose from, quite an improvement over the 4 colors in single hi-res. Because the horizontal resolution in double hi-res is 560 dots and we use 4 for each color, the resolution in double hi-res color is 560/4 = 140 or the same as in single hi-res color. But, with more colors to choose from, we can display more interesting graphics and with more apparent resolution because of color contrasts. Also, as we'll see, there is no limitation to color combinations within lines as there is in single hi-res.

Because the high bit of the shape byte is not plotted, this presents a problem when we want to repeat a particular dot pattern on the screen, which we would do, for example, in plotting a line of a single color. Suppose we want to display a dark blue line. The repeated dot pattern we want on the screen is 1 0 0 0. If we plot shape byte #$11 in AUX1, we will get the desired pattern, but if #$11 is also plotted in the next byte, MAIN1, see what happens:

```
      AUX1          MAIN1
      #$11    |     #$11     |
 1 0 0 0 1 0 0 | 1 0 0 0 1 0 0 |
```

Obviously, the desired pattern is not repeated. The pattern is repeated, however, by plotting #$22 in MAIN1 and to continue the pattern, we would plot #$44 in AUX2 and #$08 (or #$88) in MAIN2, the next two positions over.

```
      AUX1           MAIN1            AUX2            MAIN2
      #$11     |     #$22      |      #$44     |      #$08
 1 0 0 0 1 0 0 | 0 1 0 0 0 1 0 | 0 0 1 0 0 0 1 | 0 0 0 1 0 0 0
```

After these 4 bytes, the pattern repeats itself, starting with #$11 in AUX3, #$22 in MAIN3, etc. If we were to plot a dark blue line from AUX2, for example, the bytes would be #$44, #$08, #$11, #$22, #$44, etc. Thus, each color has its own sequence of 4 bytes, the particular starting byte required depending on the distance from the left screen border. The dot pattern and the 4-byte sequence for each of the 16 colors is shown in Table 12-1.

Table 12-1

Color	*Bit Pattern*	*AUX1*	*MAIN1*	*AUX2*	*MAIN2*
Black	0 0 0 0	#$00	#$00	#$00	#$00
Magenta	0 0 0 1	#$08	#$11	#$22	#$44
Brown	0 0 1 0	#$44	#$08	#$11	#$22
Orange	0 0 1 1	#$4C	#$19	#$33	#$66
Dark Green	0 1 0 0	#$22	#$44	#$08	#$11
Grey 1	0 1 0 1	#$2A	#$55	#$2A	#$55
Green	0 1 1 0	#$66	#$4C	#$19	#$33
Yellow	0 1 1 1	#$6E	#$5D	#$3B	#$77
Dark Blue	1 0 0 0	#$11	#$22	#$44	#$08
Violet	1 0 0 1	#$19	#$33	#$66	#$4C
Grey 2	1 0 1 0	#$55	#$2A	#$55	#$2A
Pink	1 0 1 1	#$5D	#$3B	#$77	#$6E
Medium Blue	1 1 0 0	#$33	#$66	#$4C	#$19
Light Blue	1 1 0 1	#$3B	#$77	#$6E	#$5D
Aqua	1 1 1 0	#$77	#$6E	#$5D	#$3B
White	1 1 1 1	#$7F	#$7F	#$7F	#$7F

This table is useful for drawing any color line anywhere on the screen. The particular starting point determines which byte is used first, then the other bytes are plotted in sequence. This is fine for a line of a single color but what if we want to plot a line with two or more colors? If the new color starts at a 4-byte boundary, we continue with the next sequence of 4 bytes for the new color. Thus, to plot a line in dark blue and magenta, with each color containing 4 bytes, the sequence would be 11,22,44,08,08,11,22,44. If the new color starts in the middle of a 4-byte sequence, we have to calculate a new byte at the color shift point by inspection. Let's say we want to plot a line containing 2 bytes of dark blue and 2 bytes of magenta. From Table 12-1 we get the values 11 and 22 for the dark blue in AUX1 and MAIN1, and 22 and 44 for the magenta in AUX2 and MAIN2. See what happens when we plot these bytes:

```
     AUX1            MAIN1             AUX2            MAIN2
     #$11     |      #$22      |       #$22     |       #$44
1 0 0 0 1 0 0 | 0 1 0 0 0 1 0 | 0 1 0 0 0 1 0 | 0 0 1 0 0 0 1
 blue    blue     blue    violet   magenta magenta magenta
```

Obviously, a different byte is required to turn that fourth position into either blue or magenta. Let's change it to a dark blue. The byte to be plotted in AUX2 is #$20:

```
     AUX1            MAIN1             AUX2            MAIN2
     #$11     |      #$22      |       #$20     |       #$44
1 0 0 0 1 0 0 | 0 1 0 0 0 1 0 | 0 0 0 0 0 1 0 | 0 0 1 0 0 0 1
 blue    blue     blue     blue    magenta magenta magenta
```

Similarly, if we want to change the fourth position to magenta, the order of bytes would be 11, 02, 22, 44. I wish I could think of some formula to make this kind of change easier but I can't—I think it just has to be done by inspection, but this is not so bad. You just decide what colors you want, inspect the dot patterns, and choose the bytes accordingly.

Note that there is no limitation for color combinations within a line—any of the 16 colors can be placed next to any other. This provides for much greater flexibility than is available in single hi-res color, over and above the larger number of colors available.

Drawing a shape in double hi-res color takes some getting used to. What you have to do is imagine that every 4 bits plot a single dot, so there are seven possible dot plots for every two screen bytes, just as in single hi-res color. The shape is plotted out, the shape bytes assembled into the usual shape tables, and the shape can then be drawn with the double hi-res routines described in the beginning of this chapter. We'll see an example in the next section.

I say "imagine" every 4 bits plot a single dot because in actuality, each bit that's "on" in the set of 4 is plotted. This is most easily seen on a monochrome monitor, but it does have implications for the color display as well. To see what this is all about, run the following program, which displays all 16 colors as lines, each directly below the other, and observe the display on both a color and monochrome monitor.

```
]PROGRAM 12-3
:ASM
                *DOUBLE HI-RES COLOR BARS
                ***********************************
                          ORG  $6000
6000: 4C 06 60            JMP  PGM
                LINE      DS   1
                BYTE      DS   1
                XCOUNT    DS   1
                GRAPHICS  =    $C050
                HIRES     =    $C057
                MIXOFF    =    $C052
                STORE     =    $C001
                AN3       =    $C05E
                COL       =    $C00D
                AUX       =    $C055
                MAIN      =    $C054
                HIGH      =    $1B
                LOW       =    $1A
6006: AD 50 C0  PGM       LDA  GRAPHICS
6009: AD 57 C0            LDA  HIRES
600C: 8D 01 C0            STA  STORE
600F: 8D 0D C0            STA  COL
6012: AD 5E C0            LDA  AN3
6015: AD 52 C0            LDA  MIXOFF
6018: AD 54 C0            LDA  MAIN
601B: 20 27 60            JSR  CLEAR        ;CLEAR MAIN P.1
601E: AD 55 C0            LDA  AUX
6021: 20 27 60            JSR  CLEAR        ;CLEAR AUX P.1
6024: 4C 41 60            JMP  START
                ************************
6027: A9 00     CLEAR     LDA  #$00         ;CLEAR SCREEN 1
6029: 85 1A               STA  LOW
602B: A9 20               LDA  #$20
602D: 85 1B               STA  HIGH
602F: A0 00     CLR1      LDY  #$00
6031: A9 00               LDA  #$00
6033: 91 1A     CLR       STA  (LOW),Y
6035: C8                  INY
6036: D0 FB               BNE  CLR
6038: E6 1B               INC  HIGH
```

```
603A: A5 1B      40            LDA  HIGH
603C: C9 40      41            CMP  #$40
603E: 90 EF      42            BLT  CLR1
6040: 60         43            RTS
                 44   ******** MAIN PROGRAM ********
6041: A9 00      45   START    LDA  #$00
6043: 8D 05 60   46            STA  XCOUNT
6046: AE 03 60   47   START1   LDX  LINE
6049: AC 04 60   48            LDY  BYTE
604C: BD D4 60   49            LDA  HI,X
604F: 85 1B      50            STA  HIGH
6051: BD 94 61   51            LDA  LO,X
6054: 85 1A      52            STA  LOW
6056: AE 05 60   53            LDX  XCOUNT
6059: AD 55 C0   54            LDA  AUX
605C: BD 94 60   55            LDA  COLOR,X
605F: 91 1A      56            STA  (LOW),Y
6061: AD 54 C0   57            LDA  MAIN
6064: BD 95 60   58            LDA  COLOR+1,X
6067: 91 1A      59            STA  (LOW),Y
6069: C8         60            INY
606A: AD 55 C0   61            LDA  AUX
606D: BD 96 60   62            LDA  COLOR+2,X
6070: 91 1A      63            STA  (LOW),Y
6072: AD 54 C0   64            LDA  MAIN
6075: BD 97 60   65            LDA  COLOR+3,X
6078: 91 1A      66            STA  (LOW),Y
607A: EE 05 60   67            INC  XCOUNT
607D: EE 05 60   68            INC  XCOUNT
6080: EE 05 60   69            INC  XCOUNT
6083: EE 05 60   70            INC  XCOUNT
6086: EE 03 60   71            INC  LINE
6089: EE 03 60   72            INC  LINE
608C: AD 03 60   73            LDA  LINE
608F: C9 1F      74            CMP  #$1F
6091: 90 B3      75            BLT  START1
6093: 60         76            RTS
6094: 00 00 00   77   COLOR    HEX  00000000
6097: 00
6098: 08 11 22   78            HEX  08112244
609B: 44
609C: 44 08 11   79            HEX  44081122
609F: 22
60A0: 4C 19 33   80            HEX  4C193366
60A3: 66
60A4: 22 44 08   81            HEX  22440811
60A7: 11
60A8: 2A 55 2A   82            HEX  2A552A55
60AB: 55
60AC: 66 4C 19   83            HEX  664C1933
60AF: 33
60B0: 6E 5D 3B   84            HEX  6E5D3B77
60B3: 77
60B4: 11 22 44   85            HEX  11224408
60B7: 08
60B8: 19 33 66   86            HEX  1933664C
60BB: 4C
60BC: 55 2A 55   87            HEX  552A552A
60BF: 2A
60C0: 5D 3B 77   88            HEX  5D3B776E
60C3: 6E
```

```
60C4: 33 66 4C  89          HEX  33664C19
60C7: 19
60C8: 3B 77 6E  90          HEX  3B776E5D
60CB: 5D
60CC: 77 6E 5D  91          HEX  776E5D3B
60CF: 3B
60D0: 7F 7F 7F  92          HEX  7F7F7F7F
60D3: 7F
                      HI
                      LO

596 bytes

Symbol table - numerical order:

   LOW      =$1A     HIGH     =$1B     LINE     =$6003    BYTE     =$6004
   XCOUNT   =$6005   PGM      =$6006   CLEAR    =$6027    CLR1     =$602F
   CLR      =$6033   START    =$6041   START1   =$6046    COLOR    =$6094
   HI       =$60D4   LO       =$6194   STORE    =$C001    COL      =$C00D
   GRAPHICS=$C050    MIXOFF   =$C052   MAIN     =$C054    AUX      =$C055
   HIRES    =$C057   AN3      =$C05E
```

On a monochrome monitor, preferably with the aid of a magnifying glass, you would observe the following dot patterns, but of course much closer than shown:

```
Magenta     0001      .    .    .    .
Brown       0010     .    .    .    .
Orange      0011     ..   ..   ..   ..
Dark Blue   1000    .    .    .    .
Aqua        1110    ...  ...  ...  ...
```


This is why double hi-res colors are distinguishable on a monochrome monitor—they all have a different dot pattern. On a color monitor, at least on mine, the individual dots are not seen; continuous color lines are.

There is a problem, however, in the color display. Look at the color monitor. Although each line is plotted starting from the first screen position (AUX1), not all line up exactly. The most extreme example is magenta and dark blue as you might suspect, because the dot patterns are 0001 and 1000; i.e., the "on" bits are at opposite ends of the 4-bit group. Other color combinations also have this alignment problem to a degree depending on the particular dot patterns—the closer the "on" bits are to each other, the lesser the problem. Thus, the 4-bit pattern not only selects a particular color, but also changes slightly exactly where the color is drawn. This presents the only limitation I can think of regarding double hi-res color combinations. If you want lines to align themselves closely, there are certain color combinations that should not be used. Thankfully, most combinations result in only a minor misalignment, so this is not a big problem but is one you should be aware of when designing your shapes.

ANIMATING DOUBLE HI-RES COLOR SHAPES

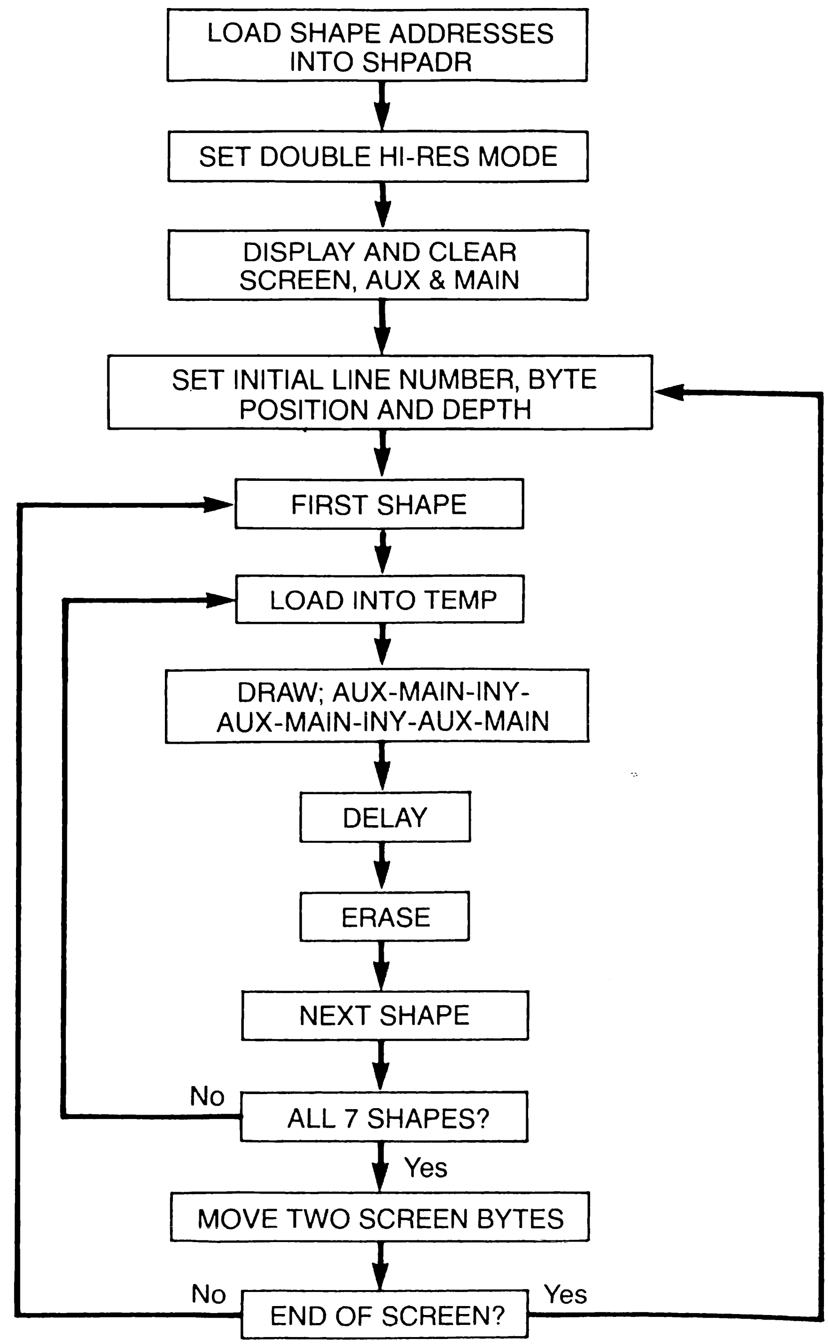
Is there a problem with vertical animation? Boo and hiss to those who answer yes. As there is no change in column assignments, the shape is just drawn once and moved up or down by changing screen line positions. Is there a problem with horizontal animation? Does Apple make computers?

Let's consider a program (Program 12-4) that moves a single dark blue line across the screen. The line length is just 2 bytes, so the first shape, at the left screen border, contains bytes #$11 and #$22 from Table 12-1. The line could be moved in whole screen-byte intervals, but this makes for rather jerky movement, so we'll use half screen-byte jumps. As with other types of horizontal movement, here too we use seven shape tables, but the particular bytes required cannot be taken from Table 12-1 except for the first shape. To illustrate this, let's look at the seven shape tables and see what bytes are required to obtain the desired dot pattern.

	AUX1	MAIN1	AUX2	MAIN2	AUX3	MAIN3
	#$11	#$22	#$00	#$00	#$00	#$00
Shape 1	1000100	0100010	0000000	0000000	0000000	0000000
	#$10	#$22	#$04	#$00	#$00	#$00
Shape 2	0000100	0100010	0010000	0000000	0000000	0000000
	#$00	#$22	#$44	#$00	#$00	#$00
Shape 3	0000000	0100010	0010001	0000000	0000000	0000000
	#$00	#$20	#$44	#$08	#$00	#$00
Shape 4	0000000	0000010	0010001	0001000	0000000	0000000
	#$00	#$00	#$44	#$08	#$01	#$00
Shape 5	0000000	0000000	0010001	0001000	1000000	0000000
	#$00	#$00	#$40	#$08	#$11	#$00
Shape 6	0000000	0000000	0000001	0001000	1000100	0000000
	#$00	#$00	#$00	#$08	#$11	#$02
Shape 7	0000000	0000000	0000000	0001000	1000100	0100000

Once the seven shapes are drawn, the pattern is repeated, but with the first shape now drawn 2 screen bytes over, i.e., #$11 in AUX3 and #$22 in MAIN3, etc. Thus, in the MAIN PROGRAM, we do INC BYTE twice after each seven shapes. Each shape table consists of 6 bytes and the draw routine is A-M-INY-A-M-INY-A-M—2 bytes over—A-M-INY-A-M-INY-A-M, etc. The remainder of the program needs no further explanation, except to remind you that it can easily be adapted to multiple line shapes by extending the shape tables and modifying TEMP and the load shape routine accordingly. Thus, to draw two blue lines of the same length, one under the other, the shape 1 table would be 11, 22, 00, 00, 00, 00, 11, 22, 00, 00, 00, 00, TEMP would be changed to DS 12, and in the

LOADSHP subroutine, the CPY #$06 would be changed to CPY #$0C to take into account the shape tables now containing 12 bytes instead of 6.




```
]PROGRAM 12-4
:ASM
                  *DOUBLE HI-RES COLOR * 1 BLUE LINE
                  ************************************
                  ************************************
                  ************************************
                           ORG   $6000
6000: 4C 1E 60             JMP   PGM
                  LINE     DS    1
                  LINEA    DS    1
                  BYTE     DS    1
```

```
                DEPTH    DS    1
                XCOUNT   DS    1
                SHPNO    DS    1
                TEMP     DS    6
                DELAY    DS    1
                GRAPHICS =     $C050
                HIRES    =     $C057
                MIXOFF   =     $C052
                STORE    =     $C001
                AN3      =     $C05E
                COL      =     $C00D
                AUX      =     $C055
                MAIN     =     $C054
                HIGH     =     $1B
                LOW      =     $1A
                WAIT     =     $FCA8
                *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
6010: 37        SHPADR   DFB   #<SHAPE1
6011: 61                 DFB   #>SHAPE1
6012: 3D                 DFB   #<SHAPE2
6013: 61                 DFB   #>SHAPE2
6014: 43                 DFB   #<SHAPE3
6015: 61                 DFB   #>SHAPE3
6016: 49                 DFB   #<SHAPE4
6017: 61                 DFB   #>SHAPE4
6018: 4F                 DFB   #<SHAPE5
6019: 61                 DFB   #>SHAPE5
601A: 55                 DFB   #<SHAPE6
601B: 61                 DFB   #>SHAPE6
601C: 5B                 DFB   #<SHAPE7
601D: 61                 DFB   #>SHAPE7
601E: AD 50 C0  PGM      LDA   GRAPHICS
6021: AD 57 C0           LDA   HIRES
6024: 8D 01 C0           STA   STORE
6027: 8D 0D C0           STA   COL
602A: AD 5E C0           LDA   AN3
602D: AD 52 C0           LDA   MIXOFF
6030: A9 70              LDA   #$70
6032: 8D 0F 60           STA   DELAY
6035: AD 54 C0           LDA   MAIN
6038: 20 44 60           JSR   CLEAR       ;CLEAR MAIN P.1
603B: AD 55 C0           LDA   AUX
603E: 20 44 60           JSR   CLEAR       ;CLEAR AUX P.1
6041: 4C 5E 60           JMP   START
                *************************
6044: A9 00     CLEAR    LDA   #$00        ;CLEAR SCREEN 1
6046: 85 1A              STA   LOW
6048: A9 20              LDA   #$20
604A: 85 1B              STA   HIGH
604C: A0 00     CLR1     LDY   #$00
604E: A9 00              LDA   #$00
6050: 91 1A     CLR      STA   (LOW),Y
6052: C8                 INY
6053: D0 FB              BNE   CLR
6055: E6 1B              INC   HIGH
6057: A5 1B              LDA   HIGH
6059: C9 40              CMP   #$40
605B: 90 EF              BLT   CLR1
605D: 60                 RTS
                ******** MAIN PROGRAM ********
605E: 20 8F 60  START    JSR   INITIAL
```

```
6061: A9 00       START1    LDA  #$00
6063: 8D 08 60              STA  SHPNO
6066: 20 A1 60    START2    JSR  LOADSHP
6069: 20 BD 60              JSR  DRAW
606C: AD 0F 60              LDA  DELAY
606F: 20 A8 FC              JSR  WAIT
6072: 20 BD 60              JSR  DRAW
6075: EE 08 60              INC  SHPNO
6078: AD 08 60              LDA  SHPNO
607B: C9 07                 CMP  #$07
607D: 90 E7                 BLT  START2
607F: EE 05 60              INC  BYTE
6082: EE 05 60              INC  BYTE
6085: AD 05 60              LDA  BYTE
6088: C9 26                 CMP  #$26
608A: 90 D5                 BLT  START1
608C: 4C 5E 60              JMP  START
                  ******* SUBROUTINES *******
608F: A9 00       INITIAL   LDA  #$00
6091: 8D 05 60              STA  BYTE
6094: 8D 03 60              STA  LINE
6097: 8D 04 60              STA  LINEA
609A: 18                    CLC
609B: 69 01                 ADC  #$01
609D: 8D 06 60              STA  DEPTH
60A0: 60                    RTS
                  ** LOAD SHAPE TABLE INTO TEMP **
60A1: AD 08 60    LOADSHP   LDA  SHPNO
60A4: 0A                    ASL
60A5: AA                    TAX
60A6: BD 10 60              LDA  SHPADR,X
60A9: 85 1A                 STA  LOW
60AB: BD 11 60              LDA  SHPADR+1,X
60AE: 85 1B                 STA  HIGH
60B0: A0 00                 LDY  #$00
60B2: B1 1A       LOADSHP1  LDA  (LOW),Y
60B4: 99 09 60              STA  TEMP,Y
60B7: C8                    INY
60B8: C0 06                 CPY  #$06
60BA: 90 F6                 BLT  LOADSHP1
60BC: 60                    RTS
                  ******************************
60BD: A9 00       DRAW      LDA  #$00
60BF: 8D 07 60              STA  XCOUNT
60C2: AC 05 60    DRAW1     LDY  BYTE
60C5: AE 03 60              LDX  LINE
60C8: BD 61 61              LDA  HI,X
60CB: 85 1B                 STA  HIGH
60CD: BD 21 62              LDA  LO,X
60D0: 85 1A                 STA  LOW
60D2: AE 07 60              LDX  XCOUNT
60D5: AD 55 C0              LDA  AUX
60D8: B1 1A                 LDA  (LOW),Y
60DA: 5D 09 60              EOR  TEMP,X
60DD: 91 1A                 STA  (LOW),Y
60DF: AD 54 C0              LDA  MAIN
60E2: B1 1A                 LDA  (LOW),Y
60E4: 5D 0A 60              EOR  TEMP+1,X
60E7: 91 1A                 STA  (LOW),Y
60E9: C8                    INY
60EA: AD 55 C0              LDA  AUX
```

```
60ED: B1 1A                LDA  (LOW),Y
60EF: 5D 0B 60             EOR  TEMP+2,X
60F2: 91 1A                STA  (LOW),Y
60F4: AD 54 C0             LDA  MAIN
60F7: B1 1A                LDA  (LOW),Y
60F9: 5D 0C 60             EOR  TEMP+3,X
60FC: 91 1A                STA  (LOW),Y
60FE: C8                   INY
60FF: AD 55 C0             LDA  AUX
6102: B1 1A                LDA  (LOW),Y
6104: 5D 0D 60             EOR  TEMP+4,X
6107: 91 1A                STA  (LOW),Y
6109: AD 54 C0             LDA  MAIN
610C: B1 1A                LDA  (LOW),Y
610E: 5D 0E 60             EOR  TEMP+5,X
6111: 91 1A                STA  (LOW),Y
6113: EE 07 60             INC  XCOUNT
6116: EE 07 60             INC  XCOUNT
6119: EE 07 60             INC  XCOUNT
611C: EE 07 60             INC  XCOUNT
611F: EE 07 60             INC  XCOUNT
6122: EE 07 60             INC  XCOUNT
6125: EE 03 60             INC  LINE
6128: AD 03 60             LDA  LINE
612B: CD 06 60             CMP  DEPTH
612E: 90 92                BLT  DRAW1
6130: AD 04 60             LDA  LINEA
6133: 8D 03 60             STA  LINE
6136: 60                   RTS
6137: 11 22 00      SHAPE1  HEX  112200000000
613A: 00 00 00
613D: 10 22 04      SHAPE2  HEX  102204000000
6140: 00 00 00
6143: 00 22 44      SHAPE3  HEX  002244000000
6146: 00 00 00
6149: 00 20 44      SHAPE4  HEX  002044080000
614C: 08 00 00
614F: 00 00 44      SHAPE5  HEX  000044080100
6152: 08 01 00
6155: 00 00 40      SHAPE6  HEX  000040081100
6158: 08 11 00
615B: 00 00 00      SHAPE7  HEX  000000081102
615E: 08 11 02
                    HI
                    LO

737 bytes

Symbol table - numerical order:

   LOW       =$1A      HIGH     =$1B      LINE     =$6003    LINEA    =$6004
   BYTE      =$6005    DEPTH    =$6006    XCOUNT   =$6007    SHPNO    =$6008
   TEMP      =$6009    DELAY    =$600F    SHPADR   =$6010    PGM      =$601E
   CLEAR     =$6044    CLR1     =$604C    CLR      =$6050    START    =$605E
   START1    =$6061    START2   =$6066    INITIAL  =$608F    LOADSHP  =$60A1
   LOADSHP1=$60B2      DRAW     =$60BD    DRAW1    =$60C2    SHAPE1   =$6137
   SHAPE2    =$613D    SHAPE3   =$6143    SHAPE4   =$6149    SHAPE5   =$614F
   SHAPE6    =$6155    SHAPE7   =$615B    HI       =$6161    LO       =$6221
   STORE     =$C001    COL      =$C00D    GRAPHICS=$C050     MIXOFF   =$C052
   MAIN      =$C054    AUX      =$C055    HIRES    =$C057    AN3      =$C05E
   WAIT      =$FCA8
```

This is a simple program, but it illustrates the basic principles of horizontal animation in double hi-res color. For more complicated shapes with multiple colors, all one has to do is map out the shape to get the proper shape bytes and to design the proper draw routine. To see double hi-res color in action, run the following program, which moves a wildly colored spaceship vertically (I don't have an extra month to draw the seven shapes for horizontal movement).

```
]PROGRAM 12-5
:ASM
                 ** DOUBLE HI-RES COLOR * VERTICAL SPACESHIP
                          ORG   $6000
6000: 4C 09 60            JMP   PGM
                 LINE     DS    1
                 LINEA    DS    1
                 BYTE     DS    1
                 DEPTH    DS    1
                 XCOUNT   DS    1
                 DELAY    DS    1
                 GRAPHICS =     $C050
                 MIXOFF   =     $C052
                 HIRES    =     $C057
                 AN3      =     $C05E
                 COL80    =     $C00D
                 STORE80  =     $C001
                 AUX      =     $C055
                 MAIN     =     $C054
                 HIGH     =     $1B
                 LOW      =     $1A
                 WAIT     =     $FCA8
6009: AD 50 C0   PGM      LDA   GRAPHICS
600C: AD 52 C0            LDA   MIXOFF
600F: AD 57 C0            LDA   HIRES
6012: 8D 01 C0            STA   STORE80
6015: 8D 0D C0            STA   COL80
6018: AD 5E C0            LDA   AN3
601B: AD 54 C0            LDA   MAIN
601E: 20 2A 60            JSR   CLEAR          ;CLEAR MAIN SCREEN
6021: AD 55 C0            LDA   AUX
6024: 20 2A 60            JSR   CLEAR          ;CLEAR AUX SCREEN
6027: 4C 44 60            JMP   DE
602A: A9 00      CLEAR    LDA   #00            ;CLEAR SCREEN SUBROUTINE
602C: 85 1A               STA   LOW
602E: A9 20               LDA   #$20
6030: 85 1B               STA   HIGH
6032: A0 00      CLR1     LDY   #00
6034: A9 00               LDA   #00
6036: 91 1A      CLR      STA   (LOW),Y
6038: C8                  INY
6039: D0 FB               BNE   CLR
603B: E6 1B               INC   HIGH
603D: A5 1B               LDA   HIGH
603F: C9 40               CMP   #$40
6041: 90 EF               BCC   CLR1
6043: 60                  RTS
6044: A9 60      DE       LDA   #$60           ;LOAD DELAY
6046: 8D 08 60            STA   DELAY
```

```
                   ********** MAIN PROGRAM **********
6049: 20 6B 60     START    JSR  INITIAL     ;SET INITIAL BYTE, LINE, DEPTH
604C: 20 7D 60     START1   JSR  DRAW
604F: AD 08 60              LDA  DELAY
6052: 20 A8 FC              JSR  WAIT
6055: 20 7D 60              JSR  DRAW
6058: EE 06 60              INC  DEPTH
605B: EE 04 60              INC  LINEA
605E: AD 04 60              LDA  LINEA
6061: 8D 03 60              STA  LINE
6064: C9 B8                 CMP  #$B8
6066: B0 E1                 BGE  START
6068: 4C 4C 60              JMP  START1
                   ********** SUBROUTINES **********
606B: A9 00        INITIAL  LDA  #$00
606D: 8D 05 60              STA  BYTE
6070: 8D 03 60              STA  LINE
6073: 8D 04 60              STA  LINEA
6076: 18                    CLC
6077: 69 09                 ADC  #$09        ;DEPTH OF SHAPE
6079: 8D 06 60              STA  DEPTH
607C: 60                    RTS
                   **********************
                   ** DRAW SUBROUTINE **
607D: A9 00        DRAW     LDA  #$00
607F: 8D 07 60              STA  XCOUNT
6082: AC 05 60     DRAW1    LDY  BYTE
6085: AE 03 60              LDX  LINE
6088: BD 4B 61              LDA  HI,X
608B: 85 1B                 STA  HIGH
608D: BD 0B 62              LDA  LO,X
6090: 85 1A                 STA  LOW
6092: AE 07 60              LDX  XCOUNT
6095: AD 55 C0              LDA  AUX
6098: B1 1A                 LDA  (LOW),Y
609A: 5D 03 61              EOR  SHAPE,X
609D: 91 1A                 STA  (LOW),Y
609F: AD 54 C0              LDA  MAIN
60A2: B1 1A                 LDA  (LOW),Y
60A4: 5D 04 61              EOR  SHAPE+1,X
60A7: 91 1A                 STA  (LOW),Y
60A9: C8                    INY
60AA: AD 55 C0              LDA  AUX
60AD: B1 1A                 LDA  (LOW),Y
60AF: 5D 05 61              EOR  SHAPE+2,X
60B2: 91 1A                 STA  (LOW),Y
60B4: AD 54 C0              LDA  MAIN
60B7: B1 1A                 LDA  (LOW),Y
60B9: 5D 06 61              EOR  SHAPE+3,X
60BC: 91 1A                 STA  (LOW),Y
60BE: C8                    INY
60BF: AD 55 C0              LDA  AUX
60C2: B1 1A                 LDA  (LOW),Y
60C4: 5D 07 61              EOR  SHAPE+4,X
60C7: 91 1A                 STA  (LOW),Y
60C9: AD 54 C0              LDA  MAIN
60CC: B1 1A                 LDA  (LOW),Y
60CE: 5D 08 61              EOR  SHAPE+5,X
60D1: 91 1A                 STA  (LOW),Y
60D3: C8                    INY
60D4: AD 55 C0              LDA  AUX
```

```
60D7: B1 1A                LDA  (LOW),Y
60D9: 5D 09 61             EOR  SHAPE+6,X
60DC: 91 1A                STA  (LOW),Y
60DE: AD 54 C0             LDA  MAIN
60E1: B1 1A                LDA  (LOW),Y
60E3: 5D 0A 61             EOR  SHAPE+7,X
60E6: 91 1A                STA  (LOW),Y
60E8: AD 07 60             LDA  XCOUNT
60EB: 18                   CLC
60EC: 69 08                ADC  #$08
60EE: 8D 07 60             STA  XCOUNT
60F1: EE 03 60             INC  LINE
60F4: AD 03 60             LDA  LINE
60F7: CD 06 60             CMP  DEPTH
60FA: 90 86                BLT  DRAW1
60FC: AD 04 60             LDA  LINEA
60FF: 8D 03 60             STA  LINE        ;RESET LINE FOR NEXT CYCLE
6102: 60                   RTS
6103: 00 00 40      SHAPE  HEX  0000405C3B000000
6106: 5C 3B 00 00 00
610B: 00 00 18             HEX  00001833664C0000
610E: 33 66 4C 00 00
6113: 00 32 66             HEX  0032664C19336600
6116: 4C 19 33 66 00
611B: 11 22 44             HEX  1122440811224408
611E: 08 11 22 44 08
6123: 11 10 46             HEX  111046701E027308
6126: 70 1E 02 73 08
612B: 33 66 4C             HEX  33664C1933664C19
612E: 19 33 66 4C 19
6133: 00 18 03             HEX  0018031002001100
6136: 10 02 00 11 00
613B: 00 5C 03             HEX  005C03680D002200
613E: 68 0D 00 22 00
6143: 00 10 02             HEX  001002700E003300
6146: 70 0E 00 33 00
                    HI
                    LO
```

715 bytes

Symbol table - numerical order:

LOW	=$1A	HIGH	=$1B	LINE	=$6003	LINEA	=$6004
BYTE	=$6005	DEPTH	=$6006	XCOUNT	=$6007	DELAY	=$6008
PGM	=$6009	CLEAR	=$602A	CLR1	=$6032	CLR	=$6036
DE	=$6044	START	=$6049	START1	=$604C	INITIAL	=$606B
DRAW	=$607D	DRAW1	=$6082	SHAPE	=$6103	HI	=$614B
LO	=$620B	STORE80	=$C001	COL80	=$C00D	GRAPHICS	=$C050
MIXOFF	=$C052	MAIN	=$C054	AUX	=$C055	HIRES	=$C057
AN3	=$C05E	WAIT	=$FCA8				

13

Curved and Diagonal Movement

Moving up and down and to and fro
Is easy enough as you well know,
But moving at an angle
Can cause quite a tangle
As you change each column and row.

Up until now we've only considered shapes moving either vertically or horizontally, but sometime in your career as a graphics computer programmer you might want to display other types of movements without having to tilt or rotate the monitor. The principle is easy. For vertical movement, we keep the screen byte constant and alter the line position; for horizontal movement, the line position is kept constant while the screen byte is changed (stop me if I'm going too fast). For diagonal or curved movement we change both the screen byte and line position for each draw. I told you it was easy.

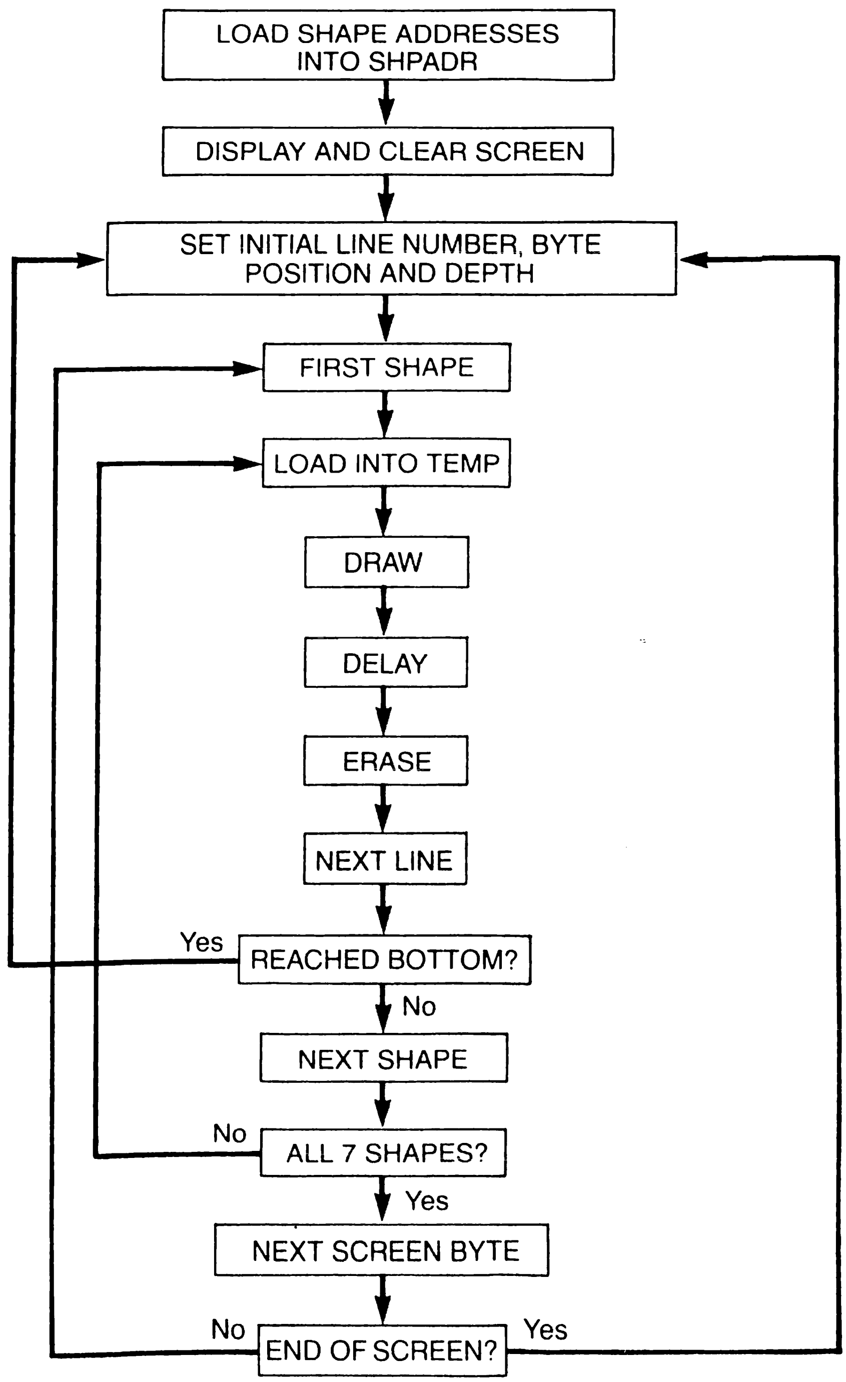
Armed with this information we can now move shapes around in any kind of meandering path but, in general, pleasant results are obtained only if shapes move in some kind of recognizable pattern, either diagonally or in a curve described by some type of simple equation (don't worry, we're not going to get into quantum mechanics or even calculus, but keep in mind that $E = mc^2$). An exception to this is when movement in all directions is controlled by a joystick or paddles as we saw in Chapter 6 (Program 6-3).

DIAGONAL MOVEMENT

For any kind of non-vertical movement, there is always a horizontal vector and so we have to use the horizontal protocol, i.e., seven preshifted shapes. To illustrate diagonal movement, we're going to use Program 5-1 as a starting point—it moves a plane shape across the screen.

For our first example, let's move the plane down one line for each horizontal 1 bit move (see Program 13-1). After each draw and erase, we do an INC LINEA (remember, we don't INC LINE because LINE is altered in the draw routine). Before going on, we test to see if we've reached the bottom of the screen (line

#$BA). If we have, we start over. If not, we continue by loading LINE with LINEA, adding the shape depth to LINE, and storing in DEPTH (DEPTH has to be changed each time the line position is changed). We then continue with the usual routine, i.e., next shape number, etc., and also test for the end of the screen. In this particular example, the plane will reach the bottom first before reaching the end of the screen because there are only 192 lines but 280 horizontal bit positions. However, we're testing for both bottom and end of screen to make the program more generally applicable.



```
]PROGRAM 13-1
:ASM
                  *1 SHAPE DIAGONAL
                  *2 BYTES WIDE, 5 LINES DEEP
                           ORG   $6000
6000: 4C 27 60             JMP   PGM
                  LINE     DS    1
                  LINEA    DS    1
                  BYTE     DS    1
                  DEPTH    DS    1
                  XCOUNT   DS    1
                  SHPNO    DS    1
                  DELAY    DS    1
                  TEMP     DS    15
                  GRAPHICS =     $C050
                  MIXOFF   =     $C052
                  HIRES    =     $C057
                  PAGE1    =     $C054
                  HIGH     =     $1B
                  LOW      =     $1A
                  WAIT     =     $FCA8
                  *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                  *CONTINUE FOR ALL 7 SHAPES
6019: 0D          SHPADR   DFB   #<SHAPE1
601A: 61                   DFB   #>SHAPE1
601B: 1C                   DFB   #<SHAPE2
601C: 61                   DFB   #>SHAPE2
601D: 2B                   DFB   #<SHAPE3
601E: 61                   DFB   #>SHAPE3
601F: 3A                   DFB   #<SHAPE4
6020: 61                   DFB   #>SHAPE4
6021: 49                   DFB   #<SHAPE5
6022: 61                   DFB   #>SHAPE5
6023: 58                   DFB   #<SHAPE6
6024: 61                   DFB   #>SHAPE6
6025: 67                   DFB   #<SHAPE7
6026: 61                   DFB   #>SHAPE7
6027: AD 50 C0    PGM      LDA   GRAPHICS    ;HIRES,P.1
602A: AD 52 C0             LDA   MIXOFF
602D: AD 57 C0             LDA   HIRES
6030: AD 54 C0             LDA   PAGE1
6033: A9 00                LDA   #$00        ;CLEAR SCREEN 1
6035: 85 1A                STA   LOW
6037: A9 20                LDA   #$20
6039: 85 1B                STA   HIGH
603B: A0 00       CLR1     LDY   #$00
603D: A9 00                LDA   #$00
603F: 91 1A       CLR      STA   (LOW),Y
6041: C8                   INY
6042: D0 FB                BNE   CLR
6044: E6 1B                INC   HIGH
6046: A5 1B                LDA   HIGH
6048: C9 40                CMP   #$40
604A: 90 EF                BLT   CLR1
604C: A9 60                LDA   #$60        ;LOAD DELAY
604E: 8D 09 60             STA   DELAY
                  ********** MAIN PROGRAM **********
6051: 20 95 60    START    JSR   INITIAL     ;SET INITIAL BYTE, LINE, DEPTH
6054: A9 00       START1   LDA   #$00        ;FIRST SHAPE NUMBER
6056: 8D 08 60             STA   SHPNO
6059: 20 A7 60    START2   JSR   LOADSHP     ;LOAD SHAPE INTO TEMP
```

```
605C: 20 C3 60             JSR  DRAW        ;DRAW
605F: AD 09 60             LDA  DELAY       ;DELAY
6062: 20 A8 FC             JSR  WAIT
6065: 20 C3 60             JSR  DRAW        ;ERASE
6068: EE 04 60             INC  LINEA
606B: AD 04 60             LDA  LINEA
606E: C9 BA                CMP  #$BA
6070: 90 03                BLT  START3
6072: 4C 51 60             JMP  START
6075: 8D 03 60   START3    STA  LINE
6078: 18                   CLC
6079: 69 05                ADC  #$05
607B: 8D 06 60             STA  DEPTH
607E: EE 08 60             INC  SHPNO       ;NEXT SHAPE NUMBER
6081: AD 08 60             LDA  SHPNO
6084: C9 07                CMP  #$07        ;FINISHED ALL 7 SHAPES?
6086: 90 D1                BLT  START2      ;IF NO, CONTINUE WITH NEXT SHAPE
6088: EE 05 60             INC  BYTE        ;IF YES, NEXT BYTE
608B: AD 05 60             LDA  BYTE
608E: C9 26                CMP  #$26        ;END OF SCREEN?
6090: 90 C2                BLT  START1      ;IF NO, CONTINUE DRAW
6092: 4C 51 60             JMP  START       ;IF YES, START OVER
                 ********** SUBROUTINES **********
6095: A9 00      INITIAL   LDA  #$00
6097: 8D 05 60             STA  BYTE
609A: 8D 03 60             STA  LINE
609D: 8D 04 60             STA  LINEA
60A0: 18                   CLC
60A1: 69 05                ADC  #$05        ;DEPTH OF SHAPE
60A3: 8D 06 60             STA  DEPTH
60A6: 60                   RTS
                 **********************
60A7: AD 08 60   LOADSHP   LDA  SHPNO       ;LOAD SHAPE INTO TEMP
60AA: 0A                   ASL
60AB: AA                   TAX
60AC: BD 19 60             LDA  SHPADR,X
60AF: 85 1A                STA  LOW
60B1: BD 1A 60             LDA  SHPADR+1,X
60B4: 85 1B                STA  HIGH
60B6: A0 00                LDY  #$00
60B8: B1 1A      LOADSHP1  LDA  (LOW),Y
60BA: 99 0A 60             STA  TEMP,Y
60BD: C8                   INY
60BE: C0 0F                CPY  #$0F
60C0: 90 F6                BLT  LOADSHP1
60C2: 60                   RTS
                 ***********************
60C3: A9 00      DRAW      LDA  #$00
60C5: 8D 07 60             STA  XCOUNT
60C8: AC 05 60   DRAW1     LDY  BYTE
60CB: AE 03 60             LDX  LINE
60CE: BD 76 61             LDA  HI,X
60D1: 85 1B                STA  HIGH
60D3: BD 36 62             LDA  LO,X
60D6: 85 1A                STA  LOW
60D8: AE 07 60             LDX  XCOUNT
60DB: B1 1A                LDA  (LOW),Y
60DD: 5D 0A 60             EOR  TEMP,X
60E0: 91 1A                STA  (LOW),Y
60E2: C8                   INY
60E3: B1 1A                LDA  (LOW),Y
```

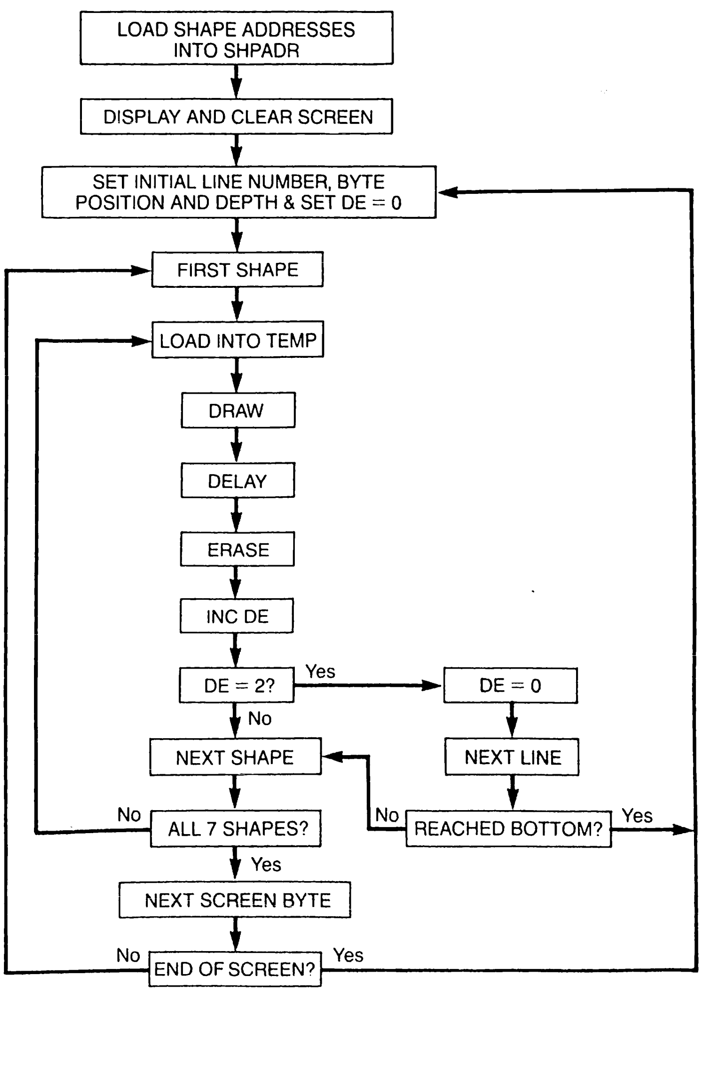
```
60E5: 5D 0B 60              EOR  TEMP+1,X
60E8: 91 1A                 STA  (LOW),Y
60EA: C8                    INY
60EB: B1 1A                 LDA  (LOW),Y
60ED: 5D 0C 60              EOR  TEMP+2,X
60F0: 91 1A                 STA  (LOW),Y
60F2: EE 07 60              INC  XCOUNT
60F5: EE 07 60              INC  XCOUNT
60F8: EE 07 60              INC  XCOUNT
60FB: EE 03 60              INC  LINE
60FE: AD 03 60              LDA  LINE
6101: CD 06 60              CMP  DEPTH
6104: 90 C2                 BLT  DRAW1
6106: AD 04 60              LDA  LINEA
6109: 8D 03 60              STA  LINE       ;RESET LINE FOR NEXT CYCLE
610C: 60                    RTS
610D: 02 00 00      SHAPE1  HEX  0200000600007E1F00 ;SHAPE TABLES
6110: 06 00 00 7E 1F 00
6116: 7E 37 00              HEX  7E37007E7F00
6119: 7E 7F 00
611C: 04 00 00      SHAPE2  HEX  0400000C00007C3F00
611F: 0C 00 00 7C 3F 00
6125: 7C 6F 00              HEX  7C6F007C7F01
6128: 7C 7F 01
612B: 08 00 00      SHAPE3  HEX  080000180000787F00
612E: 18 00 00 78 7F 00
6134: 78 5F 01              HEX  785F01787F03
6137: 78 7F 03
613A: 10 00 00      SHAPE4  HEX  100000300000707F01
613D: 30 00 00 70 7F 01
6143: 70 3F 03              HEX  703F03707F07
6146: 70 7F 07
6149: 20 00 00      SHAPE5  HEX  200000600000607F03
614C: 60 00 00 60 7F 03
6152: 60 7F 06              HEX  607F06607F0F
6155: 60 7F 0F
6158: 40 00 00      SHAPE6  HEX  400000400100407F07
615B: 40 01 00 40 7F 07
6161: 40 7F 0D              HEX  407F0D407F1F
6164: 40 7F 1F
6167: 00 01 00      SHAPE7  HEX  000100000300007F0F
616A: 00 03 00 00 7F 0F
6170: 00 7F 1B              HEX  007F1B007F3F
6173: 00 7F 3F
                    HI
                    LO
758 bytes
```

Symbol table - numerical order:

```
LOW      =$1A      HIGH    =$1B      LINE    =$6003    LINEA   =$6004
BYTE     =$6005    DEPTH   =$6006    XCOUNT  =$6007    SHPNO   =$6008
DELAY    =$6009    TEMP    =$600A    SHPADR  =$6019    PGM     =$6027
CLR1     =$603B    CLR     =$603F    START   =$6051    START1  =$6054
START2   =$6059    START3  =$6075    INITIAL =$6095    LOADSHP =$60A7
LOADSHP1=$60B8     DRAW    =$60C3    DRAW1   =$60C8    SHAPE1  =$610D
SHAPE2   =$611C    SHAPE3  =$612B    SHAPE4  =$613A    SHAPE5  =$6149
SHAPE6   =$6158    SHAPE7  =$6167    HI      =$6176    LO      =$6236
GRAPHICS=$C050     MIXOFF  =$C052    PAGE1   =$C054    HIRES   =$C057
WAIT     =$FCA8
```

We can make the plane drop at an even steeper angle simply by increasing the line positions more often than once every horizontal move. We would do INC LINEA twice, or three times, or however many we want before going on to the next draw, but keep in mind that we want to keep the line jumps to a reasonably small number to maintain smooth animation. We could, with a more complicated protocol, draw the shape at each line position instead of after each every two or three line moves, but this results in a rather noticeable jerky motion. Line jumps between draws result in a more acceptable animation as long as the distance between draws is kept small (large jumps are okay for fast moving shapes, as we'll see below).

Suppose now we want the plane to drop at a shallower angle, let's say one line for every two horizontal moves. The next program (13-2) illustrates how this is done. We set up a counter labeled DE and set it to zero in the INITIAL subroutine. After the first draw and erase, DE is incremented by 1. If DE is less than 2 (line 67), we continue drawing on the same line. After the shape has been drawn and erased two times, DE = 2 and the branch at line 68 is not taken; DE is zeroed, LINEA is incremented and, if the bottom has not yet been reached, drawing continues, now one line down. Note that each time LINEA is changed, LINE is loaded with LINEA and DEPTH is adjusted (line 76). We can easily make the shape fall in a shallower angle by changing the CMP value in line 67. Thus, if we do a CMP #$03, the shape will move three horizontal positions between each line change.



```
]PROGRAM 13-2
:ASM
                *1 SHAPE DIAGONAL  2 HORIZ. 1 VERT.
                *2 BYTES WIDE, 5 LINES DEEP
                         ORG   $6000
6000: 4C 28 60           JMP   PGM
                LINE     DS    1
                LINEA    DS    1
                BYTE     DS    1
                DEPTH    DS    1
                XCOUNT   DS    1
                SHPNO    DS    1
                DELAY    DS    1
                DE       DS    1
                TEMP     DS    15
                GRAPHICS =     $C050
                MIXOFF   =     $C052
                HIRES    =     $C057
                PAGE1    =     $C054
                HIGH     =     $1B
                LOW      =     $1A
                WAIT     =     $FCA8
                *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                *CONTINUE FOR ALL 7 SHAPES
601A: 20        SHPADR   DFB   #<SHAPE1
601B: 61                 DFB   #>SHAPE1
601C: 2F                 DFB   #<SHAPE2
601D: 61                 DFB   #>SHAPE2
601E: 3E                 DFB   #<SHAPE3
601F: 61                 DFB   #>SHAPE3
6020: 4D                 DFB   #<SHAPE4
6021: 61                 DFB   #>SHAPE4
6022: 5C                 DFB   #<SHAPE5
6023: 61                 DFB   #>SHAPE5
6024: 6B                 DFB   #<SHAPE6
6025: 61                 DFB   #>SHAPE6
6026: 7A                 DFB   #<SHAPE7
6027: 61                 DFB   #>SHAPE7
6028: AD 50 C0  PGM      LDA   GRAPHICS     ;HIRES,P.1
602B: AD 52 C0           LDA   MIXOFF
602E: AD 57 C0           LDA   HIRES
6031: AD 54 C0           LDA   PAGE1
6034: A9 00              LDA   #$00         ;CLEAR SCREEN 1
6036: 85 1A              STA   LOW
6038: A9 20              LDA   #$20
603A: 85 1B              STA   HIGH
603C: A0 00     CLR1     LDY   #$00
603E: A9 00              LDA   #$00
6040: 91 1A     CLR      STA   (LOW),Y
6042: C8                 INY
6043: D0 FB              BNE   CLR
6045: E6 1B              INC   HIGH
6047: A5 1B              LDA   HIGH
6049: C9 40              CMP   #$40
604B: 90 EF              BLT   CLR1
604D: A9 60              LDA   #$60         ;LOAD DELAY
604F: 8D 09 60           STA   DELAY
```

```
                   ********** MAIN PROGRAM **********
6052: 20 A5 60     START    JSR  INITIAL     ;SET INITIAL BYTE, LINE, DEPTH
6055: A9 00        START1   LDA  #$00        ;FIRST SHAPE NUMBER
6057: 8D 08 60              STA  SHPNO
605A: 20 BA 60     START2   JSR  LOADSHP     ;LOAD SHAPE INTO TEMP
605D: 20 D6 60              JSR  DRAW        ;DRAW
6060: AD 09 60              LDA  DELAY       ;DELAY
6063: 20 A8 FC              JSR  WAIT
6066: 20 D6 60              JSR  DRAW        ;ERASE
6069: EE 0A 60              INC  DE          ;NEXT LINE EVERY
606C: AD 0A 60              LDA  DE             TWO SHAPES
606F: C9 02                 CMP  #$02
6071: D0 1B                 BNE  START3
6073: A9 00                 LDA  #$00
6075: 8D 0A 60              STA  DE
6078: EE 04 60              INC  LINEA
607B: AD 04 60              LDA  LINEA
607E: C9 BA                 CMP  #$BA        ;TEST FOR BOTTOM
6080: 90 03                 BLT  START4
6082: 4C 52 60              JMP  START
6085: 8D 03 60     START4   STA  LINE
6088: 18                    CLC
6089: 69 05                 ADC  #$05
608B: 8D 06 60              STA  DEPTH
608E: EE 08 60     START3   INC  SHPNO       ;NEXT SHAPE NUMBER
6091: AD 08 60              LDA  SHPNO
6094: C9 07                 CMP  #$07        ;FINISHED ALL 7 SHAPES?
6096: 90 C2                 BLT  START2      ;IF NO, CONTINUE WITH NEXT SHAPE
6098: EE 05 60              INC  BYTE        ;IF YES, NEXT BYTE
609B: AD 05 60              LDA  BYTE
609E: C9 26                 CMP  #$26        ;END OF SCREEN?
60A0: 90 B3                 BLT  START1      ;IF NO, CONTINUE DRAW
60A2: 4C 52 60              JMP  START       ;IF YES, START OVER
                   ********** SUBROUTINES **********
60A5: A9 00        INITIAL  LDA  #$00
60A7: 8D 0A 60              STA  DE
60AA: 8D 05 60              STA  BYTE
60AD: 8D 03 60              STA  LINE
60B0: 8D 04 60              STA  LINEA
60B3: 18                    CLC
60B4: 69 05                 ADC  #$05        ;DEPTH OF SHAPE
60B6: 8D 06 60              STA  DEPTH
60B9: 60                    RTS
                   *********************
60BA: AD 08 60     LOADSHP  LDA  SHPNO       ;LOAD SHAPE INTO TEMP
60BD: 0A                    ASL
60BE: AA                    TAX
60BF: BD 1A 60              LDA  SHPADR,X
60C2: 85 1A                 STA  LOW
60C4: BD 1B 60              LDA  SHPADR+1,X
60C7: 85 1B                 STA  HIGH
60C9: A0 00                 LDY  #$00
60CB: B1 1A        LOADSHP1 LDA  (LOW),Y
60CD: 99 0B 60              STA  TEMP,Y
60D0: C8                    INY
60D1: C0 0F                 CPY  #$0F
60D3: 90 F6                 BLT  LOADSHP1
60D5: 60                    RTS
```

```
                 114  ***********************
60D6: A9 00      115  DRAW     LDA  #$00
60D8: 8D 07 60   116           STA  XCOUNT
60DB: AC 05 60   117  DRAW1    LDY  BYTE
60DE: AE 03 60   118           LDX  LINE
60E1: BD 89 61   119           LDA  HI,X
60E4: 85 1B      120           STA  HIGH
60E6: BD 49 62   121           LDA  LO,X
60E9: 85 1A      122           STA  LOW
60EB: AE 07 60   123           LDX  XCOUNT
60EE: B1 1A      124           LDA  (LOW),Y
60F0: 5D 0B 60   125           EOR  TEMP,X
60F3: 91 1A      126           STA  (LOW),Y
60F5: C8         127           INY
60F6: B1 1A      128           LDA  (LOW),Y
60F8: 5D 0C 60   129           EOR  TEMP+1,X
60FB: 91 1A      130           STA  (LOW),Y
60FD: C8         131           INY
60FE: B1 1A      132           LDA  (LOW),Y
6100: 5D 0D 60   133           EOR  TEMP+2,X
6103: 91 1A      134           STA  (LOW),Y
6105: EE 07 60   135           INC  XCOUNT
6108: EE 07 60   136           INC  XCOUNT
610B: EE 07 60   137           INC  XCOUNT
610E: EE 03 60   138           INC  LINE
6111: AD 03 60   139           LDA  LINE
6114: CD 06 60   140           CMP  DEPTH
6117: 90 C2      141           BLT  DRAW1
6119: AD 04 60   142           LDA  LINEA
611C: 8D 03 60   143           STA  LINE        ;RESET LINE FOR NEXT CYCLE
611F: 60         144           RTS
6120: 02 00 00   145  SHAPE1   HEX  0200000600007E1F00 ;SHAPE TABLES
6123: 06 00 00 7E 1F 00
6129: 7E 37 00   146           HEX  7E37007E7F00
612C: 7E 7F 00
612F: 04 00 00   147  SHAPE2   HEX  0400000C00007C3F00
6132: 0C 00 00 7C 3F 00
6138: 7C 6F 00   148           HEX  7C6F007C7F01
613B: 7C 7F 01
613E: 08 00 00   149  SHAPE3   HEX  080000180000787F00
6141: 18 00 00 78 7F 00
6147: 78 5F 01   150           HEX  785F01787F03
614A: 78 7F 03
614D: 10 00 00   151  SHAPE4   HEX  100000300000707F01
6150: 30 00 00 70 7F 01
6156: 70 3F 03   152           HEX  703F03707F07
6159: 70 7F 07
615C: 20 00 00   153  SHAPE5   HEX  200000600000607F03
615F: 60 00 00 60 7F 03
6165: 60 7F 06   154           HEX  607F06607F0F
6168: 60 7F 0F
616B: 40 00 00   155  SHAPE6   HEX  400000400100407F07
616E: 40 01 00 40 7F 07
6174: 40 7F 0D   156           HEX  407F0D407F1F
6177: 40 7F 1F
617A: 00 01 00   157  SHAPE7   HEX  000100000300007F0F
617D: 00 03 00 00 7F 0F
```

```
6183: 00 7F 1B   158          HEX  007F1B007F3F
6186: 00 7F 3F
                        HI
                        LO

777 bytes

Symbol table - numerical order:

   LOW     =$1A      HIGH    =$1B      LINE     =$6003     LINEA   =$6004
   BYTE    =$6005    DEPTH   =$6006    XCOUNT   =$6007     SHPNO   =$6008
   DELAY   =$6009    DE      =$600A    TEMP     =$600B     SHPADR  =$601A
   PGM     =$6028    CLR1    =$603C    CLR      =$6040     START   =$6052
   START1  =$6055    START2  =$605A    START4   =$6085     START3  =$608E
   INITIAL =$60A5    LOADSHP =$60BA    LOADSHP1=$60CB      DRAW    =$60D6
   DRAW1   =$60DB    SHAPE1  =$6120    SHAPE2   =$612F     SHAPE3  =$613E
   SHAPE4  =$614D    SHAPE5  =$615C    SHAPE6   =$616B     SHAPE7  =$617A
   HI      =$6189    LO      =$6249    GRAPHICS=$C050      MIXOFF  =$C052
   PAGE1   =$C054    HIRES   =$C057    WAIT     =$FCA8
```

CURVED MOVEMENT

In general, when moving shapes that are meant to represent some object in the real world, such as planes, bullets, bombs, or what have you, realism is effected only when the path represents how such shapes actually move. This usually means the path must follow some sort of defined curve such as a circle, parabola, etc. Of course, if you're moving a shape that looks like a snigglehof, you can twist it around any way you want, but the example I'm going to use is falling bombs, not only because it fits in well with the game program, but also because it expresses my militaristic aggression (you'll get this way, too, after a few bouts with assembly language programming).

Actually, the falling bomb example is applicable to any falling object. When something falls as a result of the force of gravity, it is constantly accelerating; that is, its vertical drop per constant horizontal displacement continually increases until it hits something or is slowed by air resistance. Let's put this in the form of equations to see how it works. We calculate new line positions as follows:

$$VX = VX + 1$$

$$LINE = LINE + VX$$

The following table illustrates how line positions change for each constant horizontal move.

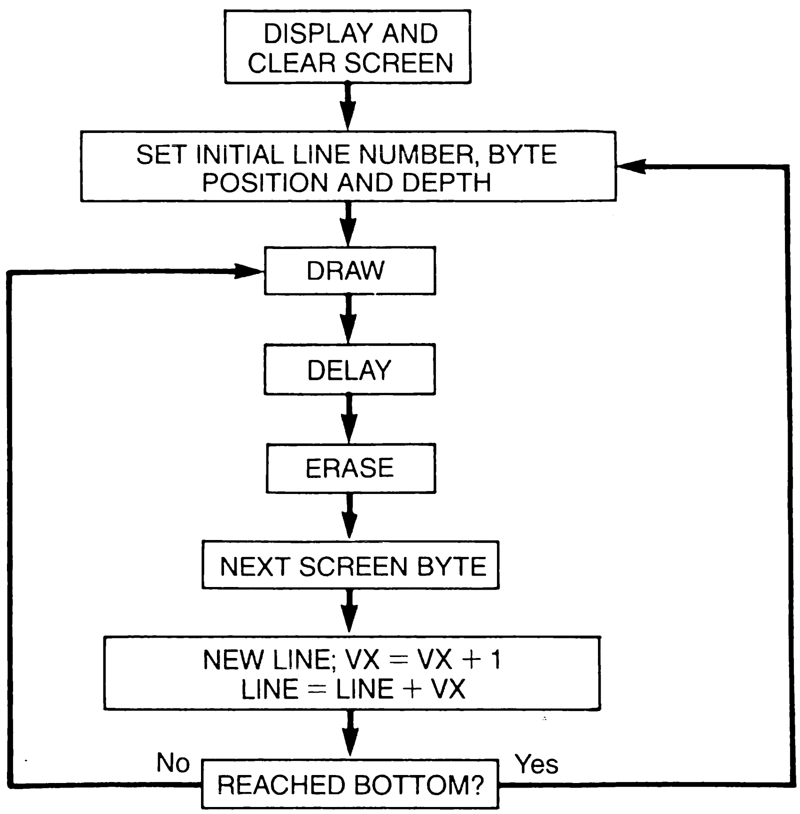
Horizontal Position	*Line*	*VX*	*New Line*
0	0	0	0
1	0	1	1
2	1	2	3
3	3	3	6
4	6	4	10
5	10	5	15
6	15	6	21

Obviously what's happening is that the distance between lines is constantly increasing by a value equal to VX and the resulting path describes a falling object exactly. Now let's see how we can put this to work in a program.

First of all, I've decided to draw the bomb at each new line position rather than continuously at each screen line; i.e., the bomb is drawn only after each line jump regardless of the distance involved. What this means is that as the bomb approaches the bottom of the screen, there will be rather large line intervals between draws, but this is just what we want. The bomb should be moving faster as it approaches the bottom and the larger line jumps provide just this illusion. Large jumps are appropriate for fast moving shapes. Look at the bullet moves in the game program—here, too, the shape is moving 1 byte (eight lines) at a time. Note also that if the bomb were drawn at every line position and not just at the new lines, the large jumps from new line to new line would be eliminated, but the result would be jerky animation and the illusion of increasing speed would be destroyed—the bomb would appear to be moving at a constant (jerky) speed all the way down. The only way to increase the apparent velocity in this case would be to shorten the delay times as the bomb falls, a tricky and unnecessary exercise, and one that wouldn't eliminate the jerky animation anyway.

The horizontal displacement of the bomb as it falls can vary from 1 bit to 1 byte or any other value you want. The displacement will not affect the acceleration illusion (this depends on the line changes), but only the steepness or shallowness of the fall. I've chosen a 1-byte move because it looks right. A 1-byte horizontal move also simplifies the program considerably because we need only one shape and not seven. The same shape is plotted at each new screen byte position. For shorter moves, we would have to use the seven preshifted shapes and change line positions after testing SHPNO for the desired values.

In the MAIN PROGRAM of Program 13-3, we draw and erase, INC BYTE, add 1 to VX, then add the value in LINE to VX and store the result in LINE and LINEA. We adjust DEPTH for the new line, test for the bottom of the screen, and then continue drawing.




```
]PROGRAM 13-3
:ASM
                   * FALLING BOMBS *
                   ****************************
                             ORG  $6000
6000: 4C 0A 60               JMP  PGM
                   XCOUNT    DS   1
                   BYTE      DS   1
                   LINE      DS   1
                   LINEA     DS   1
                   DEPTH     DS   1
                   DELAY     DS   1
                   VX        DS   1
                   GRAPHICS  =    $C050
                   MIXOFF    =    $C052
                   HIRES     =    $C057
                   PAGE1     =    $C054
                   HIGH      =    $1B
                   LOW       =    $1A
                   WAIT      =    $FCA8
600A: AD 50 C0     PGM       LDA  GRAPHICS    ;HIRES,P.1
600D: AD 52 C0               LDA  MIXOFF
6010: AD 57 C0               LDA  HIRES
6013: AD 54 C0               LDA  PAGE1
6016: A9 00                  LDA  #00         ;CLEAR SCREEN 1
```

```
6018: 85 26                STA  $26
601A: A9 20                LDA  #$20
601C: 85 27                STA  $27
601E: A0 00       CLR1     LDY  #00
6020: A9 00                LDA  #00
6022: 91 26       CLR      STA  ($26),Y
6024: C8                   INY
6025: D0 FB                BNE  CLR
6027: E6 27                INC  $27
6029: A5 27                LDA  $27
602B: C9 40                CMP  #$40
602D: 90 EF                BLT  CLR1
602F: A9 B0                LDA  #$B0         ;LOAD TIME DELAY
6031: 8D 08 60             STA  DELAY
                  ***** MAIN PROGRAM *****
6034: 20 64 60    START    JSR  INITIAL      ;SETUP BYTE,LINE & DEPTH
6037: 20 7D 60    START1   JSR  DRAW         ;DRAW SHAPE
603A: AD 08 60             LDA  DELAY        ;DELAY
603D: 20 A8 FC             JSR  WAIT
6040: 20 7D 60             JSR  DRAW         ;ERASE SHAPE
6043: EE 04 60             INC  BYTE         ;NEXT BYTE
6046: AD 09 60             LDA  VX           ;SET NEW LINE
6049: 18                   CLC
604A: 69 01                ADC  #01
604C: 8D 09 60             STA  VX
604F: 6D 05 60             ADC  LINE
6052: 8D 05 60             STA  LINE         ;NEW LINE
6055: 8D 06 60             STA  LINEA
6058: 69 03                ADC  #03          ;ADD DEPTH OF SHAPE TO NEW LINE
605A: 8D 07 60             STA  DEPTH
605D: C9 BA                CMP  #$BA         ;IS LINE AT BOTTOM OF SCREEN?
605F: B0 D3                BGE  START        ;IF YES, DRAW FROM INITIAL VALUES
6061: 4C 37 60             JMP  START1       ;IF NO, DRAW NEXT LINE, NEXT BYTE
                  ***** SUBROUTINES *****
6064: A9 00       INITIAL  LDA  #$00
6066: 8D 04 60             STA  BYTE
6069: A9 00                LDA  #00
606B: 8D 05 60             STA  LINE
606E: 8D 06 60             STA  LINEA
6071: 18                   CLC
6072: 69 03                ADC  #03
6074: 8D 07 60             STA  DEPTH
6077: A9 00                LDA  #00
6079: 8D 09 60             STA  VX
607C: 60                   RTS
                  ***********************
607D: A9 00       DRAW     LDA  #00
607F: 8D 03 60             STA  XCOUNT
6082: AC 04 60    DRAW1    LDY  BYTE
6085: AE 05 60             LDX  LINE
6088: BD B4 60             LDA  HI,X
608B: 85 1B                STA  HIGH
608D: BD 74 61             LDA  LO,X
6090: 85 1A                STA  LOW
6092: AE 03 60             LDX  XCOUNT
6095: B1 1A                LDA  (LOW),Y
6097: 5D B1 60             EOR  SHAPE,X
609A: 91 1A                STA  (LOW),Y
609C: EE 03 60             INC  XCOUNT
609F: EE 05 60             INC  LINE
60A2: AD 05 60             LDA  LINE
```

```
60A5: CD 07 60  85          CMP  DEPTH
60A8: 90 D8     86          BLT  DRAW1
60AA: AD 06 60  87          LDA  LINEA
60AD: 8D 05 60  88          STA  LINE
60B0: 60        89          RTS
60B1: 07 1E 07  90   SHAPE  HEX  071E07     ;SHAPE TABLE
                     HI
                     LO


564 bytes

Symbol table - numerical order:

   LOW      =$1A      HIGH     =$1B      XCOUNT  =$6003     BYTE     =$6004
   LINE     =$6005    LINEA    =$6006    DEPTH   =$6007     DELAY    =$6008
   VX       =$6009    PGM      =$600A    CLR1    =$601E     CLR      =$6022
   START    =$6034    START1   =$6037    INITIAL =$6064     DRAW     =$607D
   DRAW1    =$6082    SHAPE    =$60B1    HI      =$60B4     LO       =$6174
   GRAPHICS=$C050     MIXOFF   =$C052    PAGE1   =$C054     HIRES    =$C057
   WAIT     =$FCA8
```

In this program, once the bomb has reached the bottom, we start over, but we can insert any other routine here, such as an explosion, decrement score, etc. If we want an explosion—for example, when the bomb reaches the screen bottom—we need only test for the bottom line of the screen and jump to an explosion routine. If we want to test for the bomb hitting the man in the game program, we would have to include a collision test for the bomb itself, being careful to calculate just which line or group of lines the bomb would reach when hitting the man. To distinguish between hitting bottom or hitting the man, we need only determine at which line the collision occurred, as the bottom line and the man occupy different line positions.

Finally, one can add more realism to falling shapes by simulating the effect of air resistance. At some point in an object's fall, air resistance will cause the acceleration to cease and the object will fall at a constant speed. We can effect this simulation by not allowing VX to go above a certain value—when VX remains constant, the line intervals will then also be constant. A routine to accomplish this would be:

```
        LDA VX
        CMP #$04
        BGE CONT
        CLC
        ADC #$01
        STA VX
CONT    CLC
        ADC LINE
        etc.
```

14

Drawing over Backgrounds

A computer artist named Pound
Drew a woman shape nicely round.
Her repute was not well,
And just so you could tell,
He used an unsavory background.

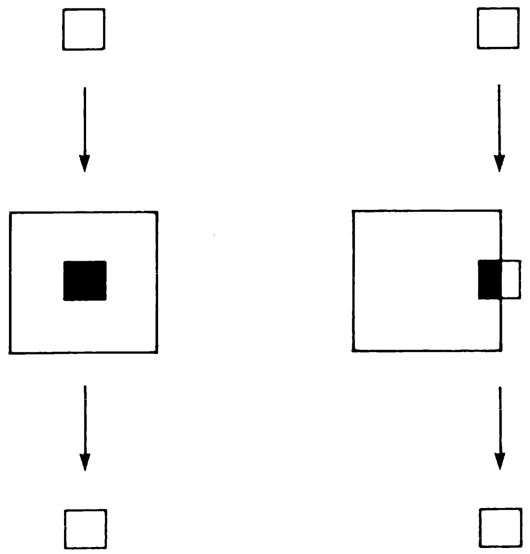
Backgrounds can enhance any program displaying hi-res graphics, not only game programs. A background can consist of stationary shapes (clouds or stars for a sky scene, for example) or moving shapes where collisions are not desired (shapes passing in the night?). Drawing a shape behind or in front of another shape can create the illusion of depth. In addition, one can use the whole screen as a background. For example, if we load the Accumulator with #$7F instead of #$00 in the clear screen routine, we can draw black shapes on a white background. Similarly, we can produce whole screen color backgrounds with the clear screen routine by LDAing with the appropriate bytes; #$55 in the even columns and #$2A in the odd columns will produce a violet screen, and so on.

WHITE SHAPES AND BACKGROUNDS

The trick to drawing over backgrounds is to have the object and the background retain their original shapes following draw-erase cycles. The easiest way to do this is to use EOR for both drawing and erasing, i.e., the usual DRAW-ERASE protocol. Let's see what happens when we EOR a shape with a background:

```
1 1 1 1 1 1 1    Background
0 0 1 1 0 0 0    EOR Shape
-------------
1 1 0 0 1 1 1    Background with shape in black
0 0 1 1 0 0 0    EOR Shape (erase)
-------------
1 1 1 1 1 1 1    Background restored
```

The result is a black shape surrounded by the white background, producing what might be called a "negative." This actually works quite well if we want the object to appear to be in front of the background as opposed to behind it. The effect is illustrated in the following figure.



To see how this looks in a program, run Program 14-1, which is the same as Program 4-1, except a box has been drawn in the path of the person shape.

```
]PROGRAM 14-1
:ASM
                ** WHITE SHAPE & BACKROUND * NEGATIVE EFFECT
                ****************************
                *SHAPE IS 1 BYTE WIDE BY 6 BYTES DEEP
                **************************
                          ORG  $6000
6000: 4C 0A 60            JMP  PGM
                XCOUNT    DS   1
                BYTE      DS   1
                LINE      DS   1
                LINEA     DS   1
                DEPTH     DS   1
                DELAY     DS   1
                SDEPTH    DS   1
                GRAPHICS  =    $C050
                MIXOFF    =    $C052
                HIRES     =    $C057
                PAGE1     =    $C054
                HIGH      =    $1B
                LOW       =    $1A
                WAIT      =    $FCA8
600A: AD 50 C0  PGM       LDA  GRAPHICS    ;HIRES,P.1
600D: AD 52 C0            LDA  MIXOFF
6010: AD 57 C0            LDA  HIRES
6013: AD 54 C0            LDA  PAGE1
6016: A9 00               LDA  #$00        ;CLEAR SCREEN 1
6018: 85 1A               STA  LOW
601A: A9 20               LDA  #$20
601C: 85 1B               STA  HIGH
```

```
601E: A0 00     CLR1    LDY  #$00
6020: A9 00             LDA  #$00
6022: 91 1A     CLR     STA  (LOW),Y
6024: C8                INY
6025: D0 FB             BNE  CLR
6027: E6 1B             INC  HIGH
6029: A5 1B             LDA  HIGH
602B: C9 40             CMP  #$40
602D: 90 EF             BLT  CLR1
602F: A9 80             LDA  #$80        ;LOAD TIME DELAY
6031: 8D 08 60          STA  DELAY
                ** DRAW WHITE BOX
6034: A9 5A             LDA  #$5A
6036: 8D 09 60          STA  SDEPTH
6039: A0 10             LDY  #$10
603B: A9 50             LDA  #$50
603D: 8D 05 60          STA  LINE
6040: AE 05 60  ST      LDX  LINE
6043: BD CC 60          LDA  HI,X
6046: 85 1B             STA  HIGH
6048: BD 8C 61          LDA  LO,X
604B: 85 1A             STA  LOW
604D: A9 7F             LDA  #$7F
604F: 91 1A             STA  (LOW),Y
6051: EE 05 60          INC  LINE
6054: AD 05 60          LDA  LINE
6057: CD 09 60          CMP  SDEPTH
605A: 90 E4             BLT  ST
                ********** MAIN PROGRAM **********
605C: 20 84 60  START   JSR  INITIAL     ;SETUP BYTE,LINE & DEPTH
605F: 20 98 60  START1  JSR  DRAW        ;DRAW SHAPE
6062: AD 08 60          LDA  DELAY       ;DELAY
6065: 20 A8 FC          JSR  WAIT
6068: AD 06 60          LDA  LINEA
606B: 8D 05 60          STA  LINE
606E: 20 98 60          JSR  DRAW        ;ERASE SHAPE
6071: EE 07 60          INC  DEPTH       ;NEXT DEPTH
6074: EE 06 60          INC  LINEA           & NEXT LINE
6077: AD 06 60          LDA  LINEA
607A: 8D 05 60          STA  LINE
607D: C9 BB             CMP  #$BB        ;IS LINE AT BOTTOM OF SCREEN?
607F: B0 DB             BGE  START       ;IF YES, DRAW FROM INITIAL VALUES
6081: 4C 5F 60          JMP  START1      ;IF NO, DRAW NEXT LINE
                ********** SUBROUTINES **********
6084: A9 10     INITIAL LDA  #$10
6086: 8D 04 60          STA  BYTE        ;SET STARTING BYTE
6089: A9 00             LDA  #$00
608B: 8D 05 60          STA  LINE        ;SET STARTING LINE
608E: 8D 06 60          STA  LINEA
6091: 18                CLC
6092: 69 06             ADC  #$06        ;ADD DEPTH OF SHAPE TO LINE
6094: 8D 07 60          STA  DEPTH
6097: 60                RTS
6098: A9 00     DRAW    LDA  #$00
609A: 8D 03 60          STA  XCOUNT      ;ZERO XCOUNT
609D: AC 04 60  DRAW1   LDY  BYTE        ;LOAD BYTE
60A0: AE 05 60          LDX  LINE        ;LOAD LINE
60A3: BD CC 60          LDA  HI,X        ;LOAD LINE ADDRESS INTO HIGH,LOW
60A6: 85 1B             STA  HIGH
60A8: BD 8C 61          LDA  LO,X
60AB: 85 1A             STA  LOW
```

```
60AD: AE 03 60               LDX  XCOUNT      ;LOAD X WITH XCOUNT
60B0: B1 1A                  LDA  (LOW),Y     ;GET BYTE FROM SCREEN
60B2: 5D C6 60               EOR  SHAPE,X     ;EOR BYTE FROM SHAPE ADDRESS+X
60B5: 91 1A                  STA  (LOW),Y     ;PLOT BYTE
60B7: EE 03 60               INC  XCOUNT
60BA: EE 05 60               INC  LINE        ;NEXT LINE
60BD: AD 05 60               LDA  LINE
60C0: CD 07 60               CMP  DEPTH       ;FINISH SHAPE?
60C3: 90 D8                  BLT  DRAW1       ;IF NO, DRAW NEXT LINE
60C5: 60                     RTS              ;IF YES, NEXT DRAW CYCLE
60C6: 08 3E 5D      SHAPE    HEX  083E5D1C1422 ;SHAPE TABLE
60C9: 1C 14 22
                    HI
                    LO

588 bytes

Symbol table - numerical order:

  LOW     =$1A      HIGH     =$1B      XCOUNT  =$6003    BYTE     =$6004
  LINE    =$6005    LINEA    =$6006    DEPTH   =$6007    DELAY    =$6008
  SDEPTH  =$6009    PGM      =$600A    CLR1    =$601E    CLR      =$6022
  ST      =$6040    START    =$605C    START1  =$605F    INITIAL  =$6084
  DRAW    =$6098    DRAW1    =$609D    SHAPE   =$60C6    HI       =$60CC
  LO      =$618C    GRAPHICS=$C050     MIXOFF  =$C052    PAGE1    =$C054
  HIRES   =$C057    WAIT     =$FCA8
```

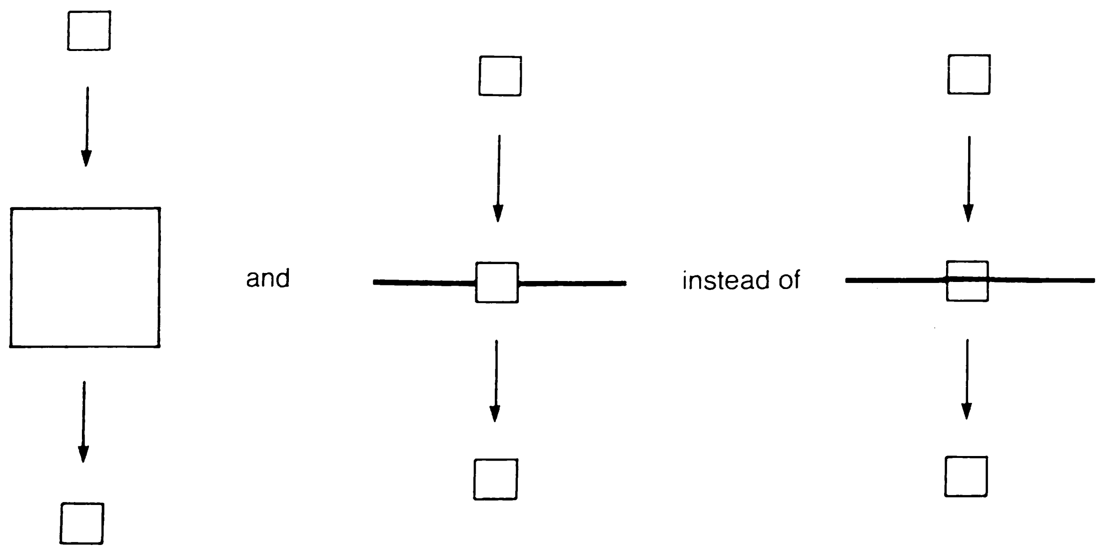
As you may have already guessed, the DRAW-DRAW protocol is inappropriate for drawing over backgrounds, because whatever background is in the screen byte will be erased by the shape byte, as there is no restoring function. Try Program 14-2, which is the same as Program 4-3 (DRAW-DRAW) except for a box in the person's path—the box is erased as the person shape passes through it.

```
]PROGRAM 14-2
:ASM
                    ** WHITE SHAPE & BACKROUND * DRAW-DRAW
                    ****************************
                    *SHAPE IS 1 BYTE WIDE BY 7 BYTES DEEP
                    ***************************
                             ORG  $6000
6000: 4C 0A 60               JMP  PGM
                    XCOUNT   DS   1
                    BYTE     DS   1
                    LINE     DS   1
                    LINEA    DS   1
                    DEPTH    DS   1
                    DELAY    DS   1
                    SDEPTH   DS   1
                    GRAPHICS =    $C050
                    MIXOFF   =    $C052
                    HIRES    =    $C057
                    PAGE1    =    $C054
                    HIGH     =    $1B
                    LOW      =    $1A
                    WAIT     =    $FCA8
600A: AD 50 C0      PGM      LDA  GRAPHICS    ;HIRES,P.1
600D: AD 52 C0               LDA  MIXOFF
```

```
6010: AD 57 C0            LDA  HIRES
6013: AD 54 C0            LDA  PAGE1
6016: A9 00               LDA  #$00       ;CLEAR SCREEN 1
6018: 85 1A               STA  LOW
601A: A9 20               LDA  #$20
601C: 85 1B               STA  HIGH
601E: A0 00      CLR1     LDY  #$00
6020: A9 00               LDA  #$00
6022: 91 1A      CLR      STA  (LOW),Y
6024: C8                  INY
6025: D0 FB               BNE  CLR
6027: E6 1B               INC  HIGH
6029: A5 1B               LDA  HIGH
602B: C9 40               CMP  #$40
602D: 90 EF               BLT  CLR1
602F: A9 80               LDA  #$80       ;LOAD TIME DELAY
6031: 8D 08 60            STA  DELAY
                 ** DRAW WHITE BOX
6034: A9 5A               LDA  #$5A
6036: 8D 09 60            STA  SDEPTH
6039: A0 10               LDY  #$10
603B: A9 50               LDA  #$50
603D: 8D 05 60            STA  LINE
6040: AE 05 60   ST       LDX  LINE
6043: BD F5 60            LDA  HI,X
6046: 85 1B               STA  HIGH
6048: BD B5 61            LDA  LO,X
604B: 85 1A               STA  LOW
604D: A9 7F               LDA  #$7F
604F: 91 1A               STA  (LOW),Y
6051: EE 05 60            INC  LINE
6054: AD 05 60            LDA  LINE
6057: CD 09 60            CMP  SDEPTH
605A: 90 E4               BLT  ST
                 ********** MAIN PROGRAM **********
605C: 20 7B 60   START    JSR  INITIAL    ;SETUP BYTE,LINE & DEPTH
605F: 20 8F 60   START1   JSR  DRAW       ;DRAW SHAPE
6062: AD 08 60            LDA  DELAY      ;DELAY
6065: 20 A8 FC            JSR  WAIT
6068: EE 07 60            INC  DEPTH      ;NEXT DEPTH
606B: EE 06 60            INC  LINEA            & NEXT LINE
606E: AD 06 60            LDA  LINEA
6071: 8D 05 60            STA  LINE
6074: C9 BA               CMP  #$BA       ;IS LINE AT BOTTOM OF SCREEN?
6076: B0 43               BGE  ERASE      ;IF YES, ERASE SHAPE, START OVER
6078: 4C 5F 60            JMP  START1     ;IF NO, DRAW NEXT LINE
                 ********** SUBROUTINES **********
607B: A9 10      INITIAL  LDA  #$10
607D: 8D 04 60            STA  BYTE       ;SET STARTING BYTE
6080: A9 00               LDA  #$00
6082: 8D 05 60            STA  LINE       ;SET STARTING LINE
6085: 8D 06 60            STA  LINEA
6088: 18                  CLC
6089: 69 07               ADC  #$07       ;ADD DEPTH OF SHAPE TO LINE
608B: 8D 07 60            STA  DEPTH
608E: 60                  RTS
608F: A9 00      DRAW     LDA  #$00
6091: 8D 03 60            STA  XCOUNT     ;ZERO XCOUNT
6094: AC 04 60   DRAW1    LDY  BYTE       ;LOAD BYTE
6097: AE 05 60            LDX  LINE       ;LOAD LINE
609A: BD F5 60            LDA  HI,X       ;LOAD LINE ADDRESS INTO HIGH,LOW
```

```
609D: 85 1B       84           STA  HIGH
609F: BD B5 61    85           LDA  LO,X
60A2: 85 1A       86           STA  LOW
60A4: AE 03 60    87           LDX  XCOUNT      ;LOAD X WITH XCOUNT
60A7: BD EE 60    88           LDA  SHAPE,X     ;LOAD SHAPE BYTE
60AA: 91 1A       89           STA  (LOW),Y     ;PLOT BYTE
60AC: EE 03 60    90           INC  XCOUNT
60AF: EE 05 60    91           INC  LINE        ;NEXT LINE
60B2: AD 05 60    92           LDA  LINE
60B5: CD 07 60    93           CMP  DEPTH       ;FINISH SHAPE?
60B8: 90 DA       94           BLT  DRAW1       ;IF NO, DRAW NEXT LINE
60BA: 60          95           RTS              ;IF YES, NEXT DRAW CYCLE
60BB: CE 05 60    96   ERASE   DEC  LINE        ;RESET LINE
60BE: A9 00       97           LDA  #$00        ;ZERO XCOUNT
60C0: 8D 03 60    98           STA  XCOUNT
60C3: AC 04 60    99   ERASE1  LDY  BYTE
60C6: AE 05 60    100          LDX  LINE
60C9: BD F5 60    101          LDA  HI,X
60CC: 85 1B       102          STA  HIGH
60CE: BD B5 61    103          LDA  LO,X
60D1: 85 1A       104          STA  LOW
60D3: AE 03 60    105          LDX  XCOUNT
60D6: B1 1A       106          LDA  (LOW),Y
60D8: 5D EE 60    107          EOR  SHAPE,X
60DB: 91 1A       108          STA  (LOW),Y     ;ERASE
60DD: EE 03 60    109          INC  XCOUNT
60E0: EE 05 60    110          INC  LINE
60E3: AD 05 60    111          LDA  LINE
60E6: CD 07 60    112          CMP  DEPTH
60E9: 90 D8       113          BLT  ERASE1
60EB: 4C 5C 60    114          JMP  START
60EE: 00 08 3E    115  SHAPE   HEX  00083E5D1C1422 ;SHAPE TABLE
60F1: 5D 1C 14 22
                       HI
                       LO

629 bytes

Symbol table - numerical order:

  LOW     =$1A      HIGH    =$1B      XCOUNT  =$6003    BYTE     =$6004
  LINE    =$6005    LINEA   =$6006    DEPTH   =$6007    DELAY    =$6008
  SDEPTH  =$6009    PGM     =$600A    CLR1    =$601E    CLR      =$6022
  ST      =$6040    START   =$605C    START1  =$605F    INITIAL  =$607B
  DRAW    =$608F    DRAW1   =$6094    ERASE   =$60BB    ERASE1   =$60C3
  SHAPE   =$60EE    HI      =$60F5    LO      =$61B5    GRAPHICS=$C050
  MIXOFF  =$C052    PAGE1   =$C054    HIRES   =$C057    WAIT     =$FCA8
```

The "negative" effect is sometimes inappropriate. For example, when we want a shape to appear to go behind the background or when the background is very simple, such as a single line, the shape should merge with the background as illustrated here.



To accomplish this, we need to use a different type of draw routine, one that uses AND and ORA.

We've seen the AND instruction before in the chapter on collisions, but it won't hurt to review it. AND compares each bit in the Accumulator with the corresponding bit of a byte, either a direct value or the contents of a memory location, and returns a value of 1 if both bits are 1. Otherwise, the result is 0. The result is stored in the Accumulator.

Example

Accumulator	0 0 1 1 0 0 1 1
AND byte	0 1 0 1 0 1 0 1
Result	0 0 0 1 0 0 0 1

ORA does the same kind of comparison, but here the result is 1 if either or both bits are 1, and 0 if both bits are 0. The result is stored in the Accumulator.

Example

Accumulator	0 0 1 1 0 0 1 1
ORA byte	0 1 0 1 0 1 0 1
Result	0 1 1 1 0 1 1 1

Let's see how we use these instructions to produce a shape merged with a background. First, we obtain a complement of the shape by EORing with #$7F. We then AND the background with the complement, and ORA the shape:

Shape	0 0 1 1 0 0 0	
EOR #$7F	1 1 1 1 1 1 1	
Result	1 1 0 0 1 1 1	
AND background	1 1 1 1 0 0 0	
Result	1 1 0 0 0 0 0	
ORA shape	0 0 1 1 0 0 0	
Result	1 1 1 1 0 0 0	Shape + Background

A problem arises when we now want to erase the shape and restore the background. If we EOR the shape, a flawed background results:

Shape + background	1 1 1 1 0 0 0
EOR shape	0 0 1 1 0 0 0
Result	1 1 0 0 0 0 0

We get around this by storing the background in a temporary location labeled BACK, and erase the shape by redrawing the stored background using the protocol LDA byte, STA screen byte. We can see how this works in the next program (Program 14-3), which is the same as Program 14-1 except for the draw and erase routines.

First, we reserve memory for the background by BACK DS 6, because the shape contains 6 bytes. In the DRAW routine, we load the screen byte by LDA (LOW),Y (line 92) and store the byte (i.e., the background) in BACK with STA BACK,X (line 93). We then continue drawing with EOR #$7F to obtain the shape complement; AND BACK,X to AND the background; ORA SHAPE,X to ORA the shape byte; and STA (LOW),Y to plot the result. X is used as the counter for the BACK "table" the same way it's used as a counter for the SHAPE table. For multiple byte shapes, we would use BACK+1,X, BACK+2,X, etc. in the same way that we use SHAPE+1,X, etc. In the MAIN PROGRAM, after the shape is drawn, we erase by calling an XDRAW routine. Here the background is restored by LDA BACK,X, STA (LOW),Y; i.e., we redraw the background directly over the merged shape + background.

```
]PROGRAM 14-3
:ASM
                 1    **WHITE SHAPE & BACKROUND * NO NEGATIVE EFFECT
                 2    ****************************
                 3    *SHAPE IS 1 BYTE WIDE BY 6 BYTES DEEP
                 4    ***************************
                 5                ORG  $6000
6000: 4C 10 60   6                JMP  PGM
                 7    XCOUNT      DS   1
                 8    BYTE        DS   1
                 9    LINE        DS   1
                 10   LINEA       DS   1
                 11   DEPTH       DS   1
                 12   DELAY       DS   1
                 13   SDEPTH      DS   1
                 14   BACK        DS   6
                 15   GRAPHICS =       $C050
                 16   MIXOFF   =       $C052
                 17   HIRES    =       $C057
                 18   PAGE1    =       $C054
                 19   HIGH     =       $1B
                 20   LOW      =       $1A
                 21   WAIT     =       $FCA8
6010: AD 50 C0   22   PGM         LDA  GRAPHICS    ;HIRES,P.1
6013: AD 52 C0   23               LDA  MIXOFF
```

```
6016: AD 57 C0             LDA  HIRES
6019: AD 54 C0             LDA  PAGE1
601C: A9 00                LDA  #$00       ;CLEAR SCREEN 1
601E: 85 1A                STA  LOW
6020: A9 20                LDA  #$20
6022: 85 1B                STA  HIGH
6024: A0 00      CLR1      LDY  #$00
6026: A9 00                LDA  #$00
6028: 91 1A      CLR       STA  (LOW),Y
602A: C8                   INY
602B: D0 FB                BNE  CLR
602D: E6 1B                INC  HIGH
602F: A5 1B                LDA  HIGH
6031: C9 40                CMP  #$40
6033: 90 EF                BLT  CLR1
6035: A9 80                LDA  #$80       ;LOAD TIME DELAY
6037: 8D 08 60             STA  DELAY
                 ** DRAW WHITE BOX
603A: A9 5A                LDA  #$5A
603C: 8D 09 60             STA  SDEPTH
603F: A0 10                LDY  #$10
6041: A9 50                LDA  #$50
6043: 8D 05 60             STA  LINE
6046: AE 05 60   ST        LDX  LINE
6049: BD 0F 61             LDA  HI,X
604C: 85 1B                STA  HIGH
604E: BD CF 61             LDA  LO,X
6051: 85 1A                STA  LOW
6053: A9 7F                LDA  #$7F
6055: 91 1A                STA  (LOW),Y
6057: EE 05 60             INC  LINE
605A: AD 05 60             LDA  LINE
605D: CD 09 60             CMP  SDEPTH
6060: 90 E4                BLT  ST
                 ********** MAIN PROGRAM **********
6062: 20 8A 60   START     JSR  INITIAL    ;SETUP BYTE,LINE & DEPTH
6065: 20 9E 60   START1    JSR  DRAW       ;DRAW SHAPE
6068: AD 08 60             LDA  DELAY      ;DELAY
606B: 20 A8 FC             JSR  WAIT
606E: AD 06 60             LDA  LINEA
6071: 8D 05 60             STA  LINE
6074: 20 D7 60             JSR  XDRAW      ;ERASE SHAPE
6077: EE 07 60             INC  DEPTH      ;NEXT DEPTH
607A: EE 06 60             INC  LINEA          & NEXT LINE
607D: AD 06 60             LDA  LINEA
6080: 8D 05 60             STA  LINE
6083: C9 BB                CMP  #$BB       ;IS LINE AT BOTTOM OF SCREEN?
6085: B0 DB                BGE  START      ;IF YES, DRAW FROM INITIAL VALUES
6087: 4C 65 60             JMP  START1     ;IF NO, DRAW NEXT LINE
                 ********** SUBROUTINES **********
608A: A9 10      INITIAL   LDA  #$10
608C: 8D 04 60             STA  BYTE       ;SET STARTING BYTE
608F: A9 00                LDA  #$00
6091: 8D 05 60             STA  LINE       ;SET STARTING LINE
6094: 8D 06 60             STA  LINEA
6097: 18                   CLC
6098: 69 06                ADC  #$06       ;ADD DEPTH OF SHAPE TO LINE
609A: 8D 07 60             STA  DEPTH
609D: 60                   RTS
609E: A9 00      DRAW      LDA  #$00
60A0: 8D 03 60             STA  XCOUNT     ;ZERO XCOUNT
```

```
60A3: AC 04 60   85   DRAW1    LDY  BYTE        ;LOAD BYTE
60A6: AE 05 60   86            LDX  LINE        ;LOAD LINE
60A9: BD 0F 61   87            LDA  HI,X        ;LOAD LINE ADDRESS INTO HIGH,LOW
60AC: 85 1B      88            STA  HIGH
60AE: BD CF 61   89            LDA  LO,X
60B1: 85 1A      90            STA  LOW
60B3: AE 03 60   91            LDX  XCOUNT      ;LOAD X WITH XCOUNT
60B6: B1 1A      92            LDA  (LOW),Y     ;GET BYTE FROM SCREEN
60B8: 9D 0A 60   93            STA  BACK,X      ;STORE BACKROUND
60BB: BD 09 61   94            LDA  SHAPE,X
60BE: 49 7F      95            EOR  #$7F
60C0: 3D 0A 60   96            AND  BACK,X
60C3: 1D 09 61   97            ORA  SHAPE,X
60C6: 91 1A      98            STA  (LOW),Y     ;PLOT
60C8: EE 03 60   99            INC  XCOUNT
60CB: EE 05 60   100           INC  LINE        ;NEXT LINE
60CE: AD 05 60   101           LDA  LINE
60D1: CD 07 60   102           CMP  DEPTH       ;FINISH SHAPE?
60D4: 90 CD      103           BLT  DRAW1       ;IF NO, DRAW NEXT LINE
60D6: 60         104           RTS              ;IF YES, NEXT DRAW CYCLE
60D7: A9 00      105  XDRAW    LDA  #$00
60D9: 8D 03 60   106           STA  XCOUNT
60DC: AE 05 60   107  XDRAW1   LDX  LINE
60DF: AC 04 60   108           LDY  BYTE
60E2: BD 0F 61   109           LDA  HI,X
60E5: 85 1B      110           STA  HIGH
60E7: BD CF 61   111           LDA  LO,X
60EA: 85 1A      112           STA  LOW
60EC: AE 03 60   113           LDX  XCOUNT
60EF: BD 0A 60   114           LDA  BACK,X      ;GET BACKROUND
60F2: 91 1A      115           STA  (LOW),Y        AND PLOT
60F4: EE 03 60   116           INC  XCOUNT
60F7: EE 05 60   117           INC  LINE
60FA: AD 05 60   118           LDA  LINE
60FD: CD 07 60   119           CMP  DEPTH
6100: 90 DA      120           BLT  XDRAW1
6102: AD 06 60   121           LDA  LINEA
6105: 8D 05 60   122           STA  LINE
6108: 60         123           RTS
6109: 08 3E 5D   124  SHAPE    HEX  083E5D1C1422 ;SHAPE TABLE
610C: 1C 14 22
                      HI
                      LO

655 bytes

Symbol table - numerical order:

   LOW      =$1A      HIGH    =$1B      XCOUNT  =$6003    BYTE    =$6004
   LINE     =$6005    LINEA   =$6006    DEPTH   =$6007    DELAY   =$6008
   SDEPTH   =$6009    BACK    =$600A    PGM     =$6010    CLR1    =$6024
   CLR      =$6028    ST      =$6046    START   =$6062    START1  =$6065
   INITIAL  =$608A    DRAW    =$609E    DRAW1   =$60A3    XDRAW   =$60D7
   XDRAW1   =$60DC    SHAPE   =$6109    HI      =$610F    LO      =$61CF
   GRAPHICS=$C050     MIXOFF  =$C052    PAGE1   =$C054    HIRES   =$C057
   WAIT     =$FCA8
```

COLOR SHAPES WITH COLOR OR WHITE BACKGROUNDS

Drawing color shapes over color or white backgrounds using the usual EOR DRAW-ERASE routine produces a variety of strange results. For example, if we plot a violet shape over a violet background or a green shape over a green background, the shape turns to black:

1 0 1 0 1 0 1	Background violet
0 0 1 0 1 0 0	EOR violet shape
1 0 0 0 0 0 1	Shape is black

Plotting blue over blue or orange over orange yields even stranger results. Here the shape not only turns to black but the background turns to the non-high-bit-set color (blue to violet or orange to green), because when the high bit is EORed, it sets to 0 (remember even though the high bit is not plotted, it is still affected by assembly language instructions):

	High	
1 0 1 0 1 0 1	1	Background blue
0 0 1 0 1 0 0	1	EOR blue shape
1 0 0 0 0 0 1	0	Black shape, violet background

If we plot alternate column colors, say a green shape over a violet background, the shape turns to white:

1 0 1 0 1 0 1	Background violet
0 0 0 1 0 1 0	EOR green shape
1 0 1 1 1 1 1	Shape is white

Similarly, if we plot a blue shape over orange, the shape also turns to white and, in addition, the background turns to the non-high-bit-set color.

If we plot color shapes over a white background using EOR, the color changes to the complement:

1 1 1 1 1 1 1	White background
1 0 1 0 1 0 1	EOR violet shape
0 1 0 1 0 1 0	Shape is green

To see the effect of all this, run the following program (Program 14-4), which draws a violet and green shape over a violet background using EOR. What you will see is that the violet part of the shape will turn to black and the green part to white as the shape passes over the background.

```
]PROGRAM 14-4
:ASM
                *COLOR SHAPE & BACKROUND WITH EOR
                ***************************
                *SHAPE IS 1 BYTE WIDE BY 6 BYTES DEEP
                **************************
                         ORG  $6000
6000: 4C 0A 60           JMP  PGM
                XCOUNT   DS   1
                BYTE     DS   1
                LINE     DS   1
                LINEA    DS   1
                DEPTH    DS   1
                DELAY    DS   1
                SDEPTH   DS   1
                GRAPHICS =    $C050
                MIXOFF   =    $C052
                HIRES    =    $C057
                PAGE1    =    $C054
                HIGH     =    $1B
                LOW      =    $1A
                WAIT     =    $FCA8
600A: AD 50 C0  PGM      LDA  GRAPHICS    ;HIRES,P.1
600D: AD 52 C0           LDA  MIXOFF
6010: AD 57 C0           LDA  HIRES
6013: AD 54 C0           LDA  PAGE1
6016: A9 00              LDA  #$00        ;CLEAR SCREEN 1
6018: 85 1A              STA  LOW
601A: A9 20              LDA  #$20
601C: 85 1B              STA  HIGH
601E: A0 00     CLR1     LDY  #$00
6020: A9 00              LDA  #$00
6022: 91 1A     CLR      STA  (LOW),Y
6024: C8                 INY
6025: D0 FB              BNE  CLR
6027: E6 1B              INC  HIGH
6029: A5 1B              LDA  HIGH
602B: C9 40              CMP  #$40
602D: 90 EF              BLT  CLR1
602F: A9 80              LDA  #$80        ;LOAD TIME DELAY
6031: 8D 08 60           STA  DELAY
                ** DRAW VIOLET BOX
6034: A9 5A              LDA  #$5A
6036: 8D 09 60           STA  SDEPTH
6039: A0 10              LDY  #$10
603B: A9 50              LDA  #$50
603D: 8D 05 60           STA  LINE
6040: AE 05 60  ST       LDX  LINE
6043: BD CC 60           LDA  HI,X
6046: 85 1B              STA  HIGH
6048: BD 8C 61           LDA  LO,X
604B: 85 1A              STA  LOW
604D: A9 55              LDA  #$55
604F: 91 1A              STA  (LOW),Y
6051: EE 05 60           INC  LINE
6054: AD 05 60           LDA  LINE
6057: CD 09 60           CMP  SDEPTH
605A: 90 E4              BLT  ST
                ********** MAIN PROGRAM **********
605C: 20 7E 60  START    JSR  INITIAL     ;SETUP BYTE,LINE & DEPTH
605F: 20 92 60  START1   JSR  DRAW        ;DRAW SHAPE
```

```
6062: AD 08 60              LDA  DELAY       ;DELAY
6065: 20 A8 FC              JSR  WAIT
6068: 20 92 60              JSR  DRAW        ;ERASE SHAPE
606B: EE 07 60              INC  DEPTH       ;NEXT DEPTH
606E: EE 06 60              INC  LINEA           & NEXT LINE
6071: AD 06 60              LDA  LINEA
6074: 8D 05 60              STA  LINE
6077: C9 BB                 CMP  #$BB        ;IS LINE AT BOTTOM OF SCREEN?
6079: B0 E1                 BGE  START       ;IF YES, DRAW FROM INITIAL VALUES
607B: 4C 5F 60              JMP  START1      ;IF NO, DRAW NEXT LINE
                 ********** SUBROUTINES **********
607E: A9 10      INITIAL    LDA  #$10
6080: 8D 04 60              STA  BYTE        ;SET STARTING BYTE
6083: A9 00                 LDA  #$00
6085: 8D 05 60              STA  LINE        ;SET STARTING LINE
6088: 8D 06 60              STA  LINEA
608B: 18                    CLC
608C: 69 06                 ADC  #$06        ;ADD DEPTH OF SHAPE TO LINE
608E: 8D 07 60              STA  DEPTH
6091: 60                    RTS
6092: A9 00      DRAW       LDA  #$00
6094: 8D 03 60              STA  XCOUNT      ;ZERO XCOUNT
6097: AC 04 60   DRAW1      LDY  BYTE        ;LOAD BYTE
609A: AE 05 60              LDX  LINE        ;LOAD LINE
609D: BD CC 60              LDA  HI,X        ;LOAD LINE ADDRESS INTO HIGH,LOW
60A0: 85 1B                 STA  HIGH
60A2: BD 8C 61              LDA  LO,X
60A5: 85 1A                 STA  LOW
60A7: AE 03 60              LDX  XCOUNT      ;LOAD X WITH XCOUNT
60AA: B1 1A                 LDA  (LOW),Y     ;GET BYTE FROM SCREEN
60AC: 5D C6 60              EOR  SHAPE,X     ;EOR BYTE FROM SHAPE ADDRESS+X
60AF: 91 1A                 STA  (LOW),Y     ;PLOT BYTE
60B1: EE 03 60              INC  XCOUNT
60B4: EE 05 60              INC  LINE        ;NEXT LINE
60B7: AD 05 60              LDA  LINE
60BA: CD 07 60              CMP  DEPTH       ;FINISH SHAPE?
60BD: 90 D8                 BLT  DRAW1       ;IF NO, DRAW NEXT LINE
60BF: AD 06 60              LDA  LINEA       ;IF YES, RESET LINE
60C2: 8D 05 60              STA  LINE          AND GO TO NEXT
60C5: 60                    RTS  DRAW       CYCLE
60C6: 14 2A 2A   SHAPE      HEX  142A2A2A1414 ;SHAPE TABLE
60C9: 2A 14 14
                 HI
                 LO

588 bytes

Symbol table - numerical order:

   LOW     =$1A        HIGH    =$1B        XCOUNT  =$6003      BYTE    =$6004
   LINE    =$6005      LINEA   =$6006      DEPTH   =$6007      DELAY   =$6008
   SDEPTH  =$6009      PGM     =$600A      CLR1    =$601E      CLR     =$6022
   ST      =$6040      START   =$605C      START1  =$605F      INITIAL =$607E
   DRAW    =$6092      DRAW1   =$6097      SHAPE   =$60C6      HI      =$60CC
   LO      =$618C      GRAPHICS=$C050      MIXOFF  =$C052      PAGE1   =$C054
   HIRES   =$C057      WAIT    =$FCA8
```

In all these cases, the background can be restored by EORing the shape byte, but clearly we need to modify the draw routine to allow the shapes to retain their colors. The technique we're going to use is a slight modification of the protocol presented in Program 14-3. We load the background byte with LDA (LOW),Y and store it in BACK with STA BACK,X. Next, we load the Accumulator, not with the shape byte but rather with a white dummy shape, and use this for the EOR #$7F and AND BACK,X. We then ORA the color shape and plot. The background is restored from BACK in the XDRAW routine. This technique is used in the following program (Program 14-5) to draw a green shape over a violet background.

```
]PROGRAM 14-5
:ASM
                     * COLOR SHAPE & BACKROUND * COLOR RETAINED
                     ****************************
                     *SHAPE IS 1 BYTE WIDE BY 6 BYTES DEEP
                     ***************************
                               ORG  $6000
6000: 4C 10 60                 JMP  PGM
                     XCOUNT    DS   1
                     BYTE      DS   1
                     LINE      DS   1
                     LINEA     DS   1
                     DEPTH     DS   1
                     DELAY     DS   1
                     SDEPTH    DS   1
                     BACK      DS   6
                     GRAPHICS  =    $C050
                     MIXOFF    =    $C052
                     HIRES     =    $C057
                     PAGE1     =    $C054
                     HIGH      =    $1B
                     LOW       =    $1A
                     WAIT      =    $FCA8
6010: AD 50 C0       PGM       LDA  GRAPHICS     ;HIRES,P.1
6013: AD 52 C0                 LDA  MIXOFF
6016: AD 57 C0                 LDA  HIRES
6019: AD 54 C0                 LDA  PAGE1
601C: A9 00                    LDA  #$00         ;CLEAR SCREEN 1
601E: 85 1A                    STA  LOW
6020: A9 20                    LDA  #$20
6022: 85 1B                    STA  HIGH
6024: A0 00          CLR1      LDY  #$00
6026: A9 00                    LDA  #$00
6028: 91 1A          CLR       STA  (LOW),Y
602A: C8                       INY
602B: D0 FB                    BNE  CLR
602D: E6 1B                    INC  HIGH
602F: A5 1B                    LDA  HIGH
6031: C9 40                    CMP  #$40
6033: 90 EF                    BLT  CLR1
6035: A9 80                    LDA  #$80         ;LOAD TIME DELAY
6037: 8D 08 60                 STA  DELAY
                     ** DRAW VIOLET BOX
603A: A9 5A                    LDA  #$5A
603C: 8D 09 60                 STA  SDEPTH
603F: A0 10                    LDY  #$10
6041: A9 50                    LDA  #$50
6043: 8D 05 60                 STA  LINE
```

```
6046: AE 05 60   ST        LDX  LINE
6049: BD 15 61             LDA  HI,X
604C: 85 1B                STA  HIGH
604E: BD D5 61             LDA  LO,X
6051: 85 1A                STA  LOW
6053: A9 55                LDA  #$55
6055: 91 1A                STA  (LOW),Y
6057: EE 05 60             INC  LINE
605A: AD 05 60             LDA  LINE
605D: CD 09 60             CMP  SDEPTH
6060: 90 E4                BLT  ST
                 ********** MAIN PROGRAM **********
6062: 20 84 60   START     JSR  INITIAL     ;SETUP BYTE,LINE & DEPTH
6065: 20 98 60   START1    JSR  DRAW        ;DRAW SHAPE
6068: AD 08 60             LDA  DELAY       ;DELAY
606B: 20 A8 FC             JSR  WAIT
606E: 20 D7 60             JSR  XDRAW       ;ERASE SHAPE
6071: EE 07 60             INC  DEPTH       ;NEXT DEPTH
6074: EE 06 60             INC  LINEA           & NEXT LINE
6077: AD 06 60             LDA  LINEA
607A: 8D 05 60             STA  LINE
607D: C9 BB                CMP  #$BB        ;IS LINE AT BOTTOM OF SCREEN?
607F: B0 E1                BGE  START       ;IF YES, DRAW FROM INITIAL VALUES
6081: 4C 65 60             JMP  START1      ;IF NO, DRAW NEXT LINE
                 ********** SUBROUTINES **********
6084: A9 10      INITIAL   LDA  #$10
6086: 8D 04 60             STA  BYTE        ;SET STARTING BYTE
6089: A9 00                LDA  #$00
608B: 8D 05 60             STA  LINE        ;SET STARTING LINE
608E: 8D 06 60             STA  LINEA
6091: 18                   CLC
6092: 69 06                ADC  #$06        ;ADD DEPTH OF SHAPE TO LINE
6094: 8D 07 60             STA  DEPTH
6097: 60                   RTS
6098: A9 00      DRAW      LDA  #$00
609A: 8D 03 60             STA  XCOUNT      ;ZERO XCOUNT
609D: AC 04 60   DRAW1     LDY  BYTE        ;LOAD BYTE
60A0: AE 05 60             LDX  LINE        ;LOAD LINE
60A3: BD 15 61             LDA  HI,X        ;LOAD LINE ADDRESS INTO HIGH,LOW
60A6: 85 1B                STA  HIGH
60A8: BD D5 61             LDA  LO,X
60AB: 85 1A                STA  LOW
60AD: AE 03 60             LDX  XCOUNT
60B0: B1 1A                LDA  (LOW),Y     ;GET SCREEN BYTE
60B2: 9D 0A 60             STA  BACK,X      ;SAVE BACKROUND
60B5: BD 0F 61             LDA  WSHAPE,X    ;LOAD WHITE SHAPE
60B8: 49 7F                EOR  #$7F
60BA: 3D 0A 60             AND  BACK,X
60BD: 1D 09 61             ORA  SHAPE,X     ;ORA COLOR SHAPE
60C0: 91 1A                STA  (LOW),Y     ;PLOT
60C2: EE 03 60             INC  XCOUNT
60C5: EE 05 60             INC  LINE        ;NEXT LINE
60C8: AD 05 60             LDA  LINE
60CB: CD 07 60             CMP  DEPTH       ;FINISH SHAPE?
60CE: 90 CD                BLT  DRAW1       ;IF NO, DRAW NEXT LINE
60D0: AD 06 60             LDA  LINEA       ;IF YES, RESET LINE
60D3: 8D 05 60             STA  LINE          AND GO TO NEXT
60D6: 60                   RTS  DRAW        CYCLE
60D7: A9 00      XDRAW     LDA  #$00
60D9: 8D 03 60             STA  XCOUNT
60DC: AE 05 60   XDRAW1    LDX  LINE
```

```
60DF: AC 04 60  108          LDY  BYTE
60E2: BD 15 61  109          LDA  HI,X
60E5: 85 1B     110          STA  HIGH
60E7: BD D5 61  111          LDA  LO,X
60EA: 85 1A     112          STA  LOW
60EC: AE 03 60  113          LDX  XCOUNT
60EF: BD 0A 60  114          LDA  BACK,X
60F2: 91 1A     115          STA  (LOW),Y
60F4: EE 03 60  116          INC  XCOUNT
60F7: EE 05 60  117          INC  LINE
60FA: AD 05 60  118          LDA  LINE
60FD: CD 07 60  119          CMP  DEPTH
6100: 90 DA     120          BLT  XDRAW1
6102: AD 06 60  121          LDA  LINEA
6105: 8D 05 60  122          STA  LINE
6108: 60        123          RTS
6109: 28 28 28  124  SHAPE   HEX  282828282828 ;SHAPE TABLE
610C: 28 28 28
610F: 7C 7C 7C  125  WSHAPE  HEX  7C7C7C7C7C7C  ;WHITE SHAPE TABLE
6112: 7C 7C 7C
                     HI
                     LO

661 bytes

Symbol table - numerical order:

  LOW     =$1A     HIGH     =$1B     XCOUNT  =$6003   BYTE    =$6004
  LINE    =$6005   LINEA    =$6006   DEPTH   =$6007   DELAY   =$6008
  SDEPTH  =$6009   BACK     =$600A   PGM     =$6010   CLR1    =$6024
  CLR     =$6028   ST       =$6046   START   =$6062   START1  =$6065
  INITIAL =$6084   DRAW     =$6098   DRAW1   =$609D   XDRAW   =$60D7
  XDRAW1  =$60DC   SHAPE    =$6109   WSHAPE  =$610F   HI      =$6115
  LO      =$61D5   GRAPHICS=$C050    MIXOFF  =$C052   PAGE1   =$C054
  HIRES   =$C057   WAIT     =$FCA8
```

Let's examine the details to see how the program works.

Violet background	1 0 1 0 1 0 1	#$55
Green shape	0 0 0 1 0 1 0	#$28
White dummy shape	0 0 1 1 1 1 1	#$7C
White shape	0 0 1 1 1 1 1	
EOR #$7F	1 1 1 1 1 1 1	
Result	1 1 0 0 0 0 0	
AND violet background	1 0 1 0 1 0 1	
Result	1 0 0 0 0 0 0	
ORA green shape	0 0 0 1 0 1 0	
Result	1 0 0 1 0 1 0	
	black	

The result is a green shape over a violet background. This is what we want—the colors are retained—but notice that the shape now has a black border. This is not a great problem. If you run Program 14-5, you'll see that the border actually sets off the shape quite nicely. Eliminating the border is really not necessary for most situations and in fact an equally pleasing effect can be achieved by changing the border to white. All that's required is changing the white dummy shape. For example:

Violet background	1 0 1 0 1 0 1	#$55
Green shape	0 0 0 1 0 1 0	#$28
White dummy shape	0 0 0 1 1 1 1	#$78
White shape	0 0 0 1 1 1 1	
EOR #$7F	1 1 1 1 1 1 1	
Result	1 1 1 0 0 0 0	
AND violet background	1 0 1 0 1 0 1	
Result	1 0 1 0 0 0 0	
ORA green shape	0 0 0 1 0 1 0	
Result	1 0 1 1 0 1 0	
	white	

This technique works for drawing any color over a white background and for any color (including white) over any other color background, unless the color combinations involve high-bit-set and high-bit-not-set colors. You can't draw a violet shape over a blue background, for example, because the plotted byte either has the high bit set or not.

15

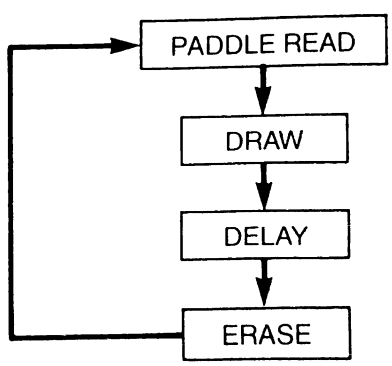
Advanced Paddle (Joystick) Routines

I'm really in a bit of a fix—
I need a limerick like a magician needs tricks.
But I'm lazy today
So I'll take the easy way—
Just read the one in Chapter 6.

The paddle routine in the game program works okay for its stated purpose, but let's see how we can use our assembly language expertise to improve on it and at the same time exercise our programming skills. First, we'll discuss how to minimize flicker by introducing a paddle movement test, and then we'll go on to a paddle-smoothing routine that prevents instantaneous movement of the paddle-controlled shape.

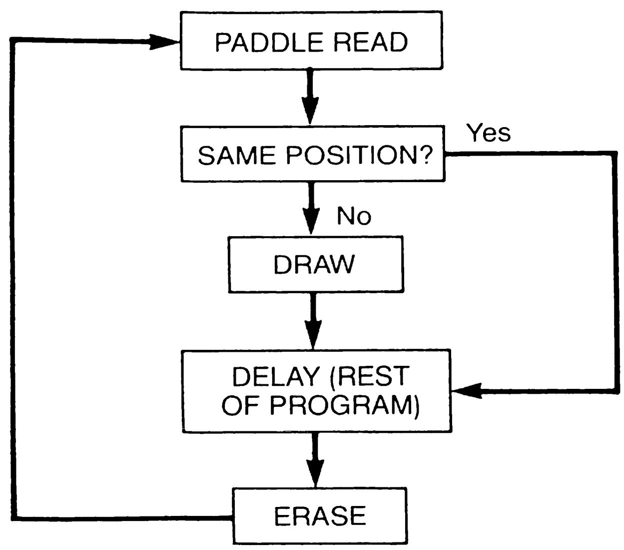
TESTING FOR NON-MOVEMENT OF PADDLE

Let's consider Program 6-1, where the vertical movement of a shape is controlled by a paddle. The overall scheme can be represented as follows:

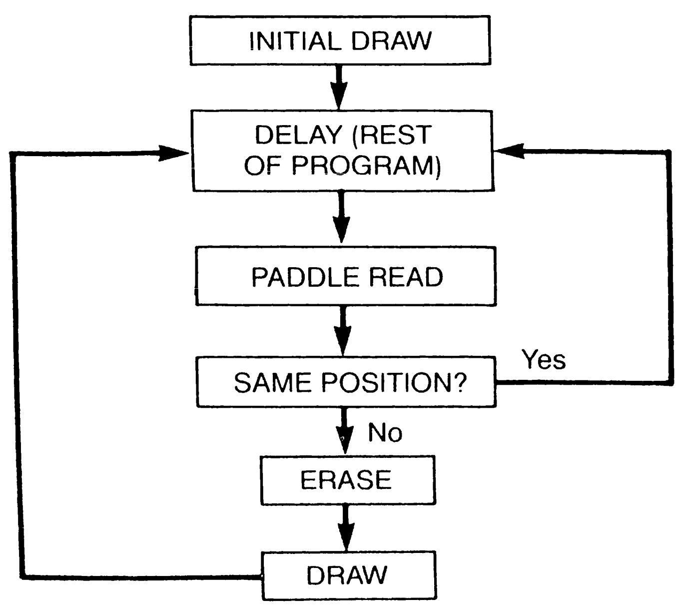


In the game program, "DELAY" can be replaced by "REST OF PROGRAM", because everything else is executed between paddle reads. Now if you look at the man shape in the game program or the shape in Program 6-1, when the shapes are stationary, flicker is evident. As mentioned before, the amount of flicker depends to a large extent on the image retention characteristics of the monitor or TV. The reason for the flickering is the delay between paddle reads; the longer the delay (or the larger the program code between reads) the greater the flickering. Note also that the flickering is noticeable only when the shape is stationary, i.e., when the paddle position is not changed, and this leads us to the solution to the problem. In the scheme diagrammed above, the shape is drawn and erased continuously, even if the paddle position stays the same. What we need to do then is introduce a test for paddle movement—if the paddle is not moved, the draw-erase cycle will be bypassed, the shape will stay on the screen at the position determined by the paddle, and flicker will be totally eliminated.

If we try to introduce this test using the scheme above, we run into trouble, because the shape is erased before each paddle read and so if the paddle isn't moved, the shape will not be displayed.



We could get around this by using a DRAW-DRAW routine, but this presents its own problems which we'll get to later. To perform the test with a DRAW-ERASE routine, we have to modify our usual draw-erase cycle to an erase-draw cycle preceded by an initial draw outside the main loop; that is:

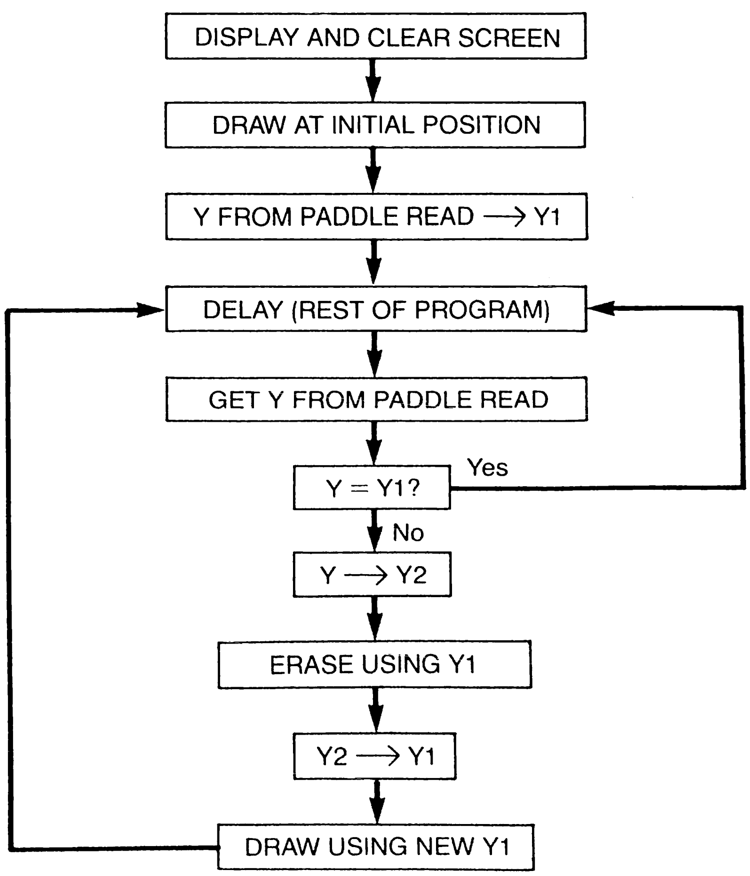


As you can see, the shape is erased and drawn only when the paddle position changes—if it stays the same, the erase-draw routine is bypassed entirely. Note that the shape is drawn, not erased, before the paddle read and test and so always stays on the screen. This scheme is incorporated into the following program (Program 15-1) which is the same as Program 6-1, except for the paddle movement test.

In the MAIN PROGRAM of Program 15-1, we draw the shape initially using a specified screen byte position (defined in the INITIAL subroutine) and a screen line specified by the PDLE subroutine; in this subroutine we also store the Y value returned from PREAD in Y1. The program then proceeds into the main loop, starting with a delay (or rest of program) and then a paddle read. The Y value returned from PREAD is compared to Y1—if equal, it means the paddle position hasn't changed and the program loops back to the delay (rest of program) without erasing and redrawing the shape. Voila, no flicker, or, as they say in French, voila, no flicker.

If Y is not equal to Y1, the paddle position has moved, so we want to erase and then redraw the shape at the new position specified by the paddle read. First, we store the Y value from PREAD temporarily in Y2 and then jump to PLOT to erase the shape using Y1 (the original Y value from the last draw). The value in Y2 is then placed in Y1 and another jump to PLOT draws the shape using the Y value from the last PREAD. In other words, Y1 is used for erasing and Y2 for drawing, then Y2 is placed in Y1 in preparation for the next cycle. After the draw, the program loops to the delay (rest of program), and so on and on and on.

Compare Programs 6-1 and 15-1. The absence of flicker is quite noticeable and quite an improvement, and would be even more so in programs with lots of code between paddle reads.



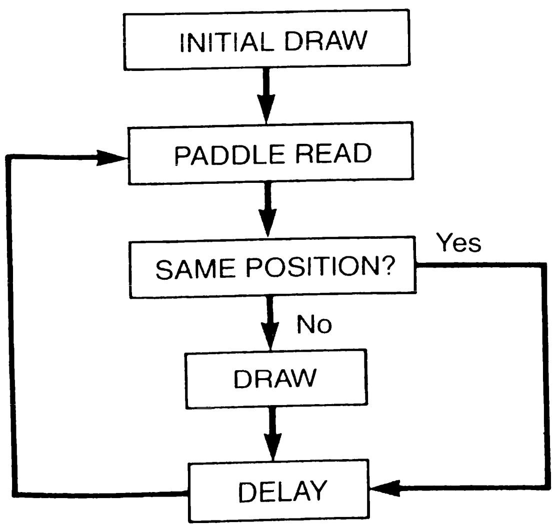
```
]PROGRAM 15-1
:ASM
                  * PADDLE MOVE TEST * VERTICAL
                  ***************************
                  *SHAPE IS 1 BYTE WIDE BY 6 BYTES DEEP
                  ***************************
                           ORG   $6000
6000: 4C 0B 60             JMP   PGM
                  XCOUNT   DS    1
                  BYTE     DS    1
                  LINE     DS    1
                  LINEA    DS    1
                  DEPTH    DS    1
                  DELAY    DS    1
                  Y1       DS    1
                  Y2       DS    1
                  GRAPHICS =     $C050
                  MIXOFF   =     $C052
                  HIRES    =     $C057
                  PAGE1    =     $C054
                  HIGH     =     $1B
                  LOW      =     $1A
                  WAIT     =     $FCA8
                  PREAD    =     $FB1E
600B: AD 50 C0    PGM      LDA   GRAPHICS    ;HIRES,P.1
600E: AD 52 C0             LDA   MIXOFF
6011: AD 57 C0             LDA   HIRES
6014: AD 54 C0             LDA   PAGE1
6017: A9 00                LDA   #$00        ;CLEAR SCREEN 1
6019: 85 1A                STA   LOW
601B: A9 20                LDA   #$20
601D: 85 1B                STA   HIGH
601F: A0 00       CLR1     LDY   #$00
6021: A9 00                LDA   #$00
6023: 91 1A       CLR      STA   (LOW),Y
6025: C8                   INY
6026: D0 FB                BNE   CLR
6028: E6 1B                INC   HIGH
602A: A5 1B                LDA   HIGH
602C: C9 40                CMP   #$40
602E: 90 EF                BLT   CLR1
6030: A9 40                LDA   #$40        ;LOAD TIME DELAY
6032: 8D 08 60             STA   DELAY
                  ********** MAIN PROGRAM **********
                  ** INITIAL DRAW **
6035: 20 76 60             JSR   INITIAL     ;SET SCREEN BYTE
6038: 20 89 60             JSR   PDLE        ;READ PADDLE 1
603B: 20 7C 60             JSR   DEP         ;SET DEPTH
603E: 20 9C 60             JSR   DRAW        ;DRAW
                  *************************
6041: AD 08 60    PROGRAM  LDA   DELAY
6044: 20 A8 FC             JSR   WAIT        ;DELAY OR REST OF PROGRAM
6047: A2 01                LDX   #$01
6049: 20 1E FB             JSR   PREAD
604C: CC 09 60             CPY   Y1          ;IF PADDLE HASN'T MOVED, DO NOT
604F: F0 F0                BEQ   PROGRAM        ERASE AND REDRAW SHAPE
```

```
6051: 8C 0A 60              STY  Y2          ;IF PADDLE HAS MOVED, STORE Y IN
6054: 20 63 60              JSR  PLOT            Y2 AND ERASE USING Y1
6057: AD 0A 60              LDA  Y2          ;TRANSFER Y2 TO Y1
605A: 8D 09 60              STA  Y1                AND
605D: 20 63 60              JSR  PLOT               DRAW
6060: 4C 41 60              JMP  PROGRAM
                  ********** SUBROUTINES **********
6063: AD 09 60    PLOT      LDA  Y1
6066: C9 BB                 CMP  #$BB
6068: 90 02                 BLT  CONT1
606A: A9 BA                 LDA  #$BA
606C: 8D 05 60    CONT1     STA  LINE
606F: 20 7C 60              JSR  DEP
6072: 20 9C 60              JSR  DRAW
6075: 60                    RTS
                  *************************
6076: A9 10       INITIAL   LDA  #$10
6078: 8D 04 60              STA  BYTE        ;SET STARTING BYTE
607B: 60                    RTS
                  ************************
607C: AD 05 60    DEP       LDA  LINE        ;SET DEPTH
607F: 8D 06 60              STA  LINEA
6082: 18                    CLC
6083: 69 06                 ADC  #$06
6085: 8D 07 60              STA  DEPTH
6088: 60                    RTS
                  ***********************
6089: A2 01       PDLE      LDX  #$01        ;READ PADDLE 1
608B: 20 1E FB              JSR  PREAD       ;0-255 IN Y
608E: 8C 09 60              STY  Y1          ;STORE Y IN Y1
6091: C0 BB                 CPY  #$BB        ;CLIP TO 0-186
6093: 90 03                 BLT  CONT
6095: A9 BA                 LDA  #$BA
6097: A8                    TAY
6098: 8C 05 60    CONT      STY  LINE        ;0-186 IN LINE
609B: 60                    RTS
                  **********************
609C: A9 00       DRAW      LDA  #$00
609E: 8D 03 60              STA  XCOUNT      ;ZERO XCOUNT
60A1: AC 04 60    DRAW1     LDY  BYTE        ;LOAD BYTE
60A4: AE 05 60              LDX  LINE        ;LOAD LINE
60A7: BD D6 60              LDA  HI,X        ;LOAD LINE ADDRESS INTO HIGH,LOW
60AA: 85 1B                 STA  HIGH
60AC: BD 96 61              LDA  LO,X
60AF: 85 1A                 STA  LOW
60B1: AE 03 60              LDX  XCOUNT      ;LOAD X WITH XCOUNT
60B4: B1 1A                 LDA  (LOW),Y     ;GET BYTE FROM SCREEN
60B6: 5D D0 60              EOR  SHAPE,X     ;EOR BYTE FROM SHAPE ADDRESS+X
60B9: 91 1A                 STA  (LOW),Y     ;PLOT BYTE
60BB: EE 03 60              INC  XCOUNT
60BE: EE 05 60              INC  LINE        ;NEXT LINE
60C1: AD 05 60              LDA  LINE
60C4: CD 07 60              CMP  DEPTH       ;FINISH SHAPE?
60C7: 90 D8                 BLT  DRAW1       ;IF NO, DRAW NEXT LINE
60C9: AD 06 60              LDA  LINEA       ;IF YES, RESET LINE AND
60CC: 8D 05 60              STA  LINE           DRAW NEXT CYCLE
60CF: 60                    RTS
```

```
60D0: 08 1C 22   112  SHAPE    HEX  081C223E227F ;SHAPE TABLE
60D3: 3E 22 7F
                      HI
                      LO


598 bytes


Symbol table - numerical order:

   LOW      =$1A       HIGH     =$1B       XCOUNT  =$6003     BYTE    =$6004
   LINE     =$6005     LINEA    =$6006     DEPTH   =$6007     DELAY   =$6008
   Y1       =$6009     Y2       =$600A     PGM     =$600B     CLR1    =$601F
   CLR      =$6023     PROGRAM  =$6041     PLOT    =$6063     CONT1   =$606C
   INITIAL  =$6076     DEP      =$607C     PDLE    =$6089     CONT    =$6098
   DRAW     =$609C     DRAW1    =$60A1     SHAPE   =$60D0     HI      =$60D6
   LO       =$6196     GRAPHICS=$C050      MIXOFF  =$C052     PAGE1   =$C054
   HIRES    =$C057     PREAD    =$FB1E     WAIT    =$FCA8
```

PADDLE-SMOOTHING ROUTINES

As mentioned above, using a DRAW-DRAW routine would simplify things somewhat, because without an erase routine, the shape will always be on the screen. For example, we could use the following scheme:



However, there is a problem with paddle routines using DRAW-DRAW. Remember that DRAW-DRAW erases by redrawing over a previous position. For vertical movement, a border of #$00's equal to the maximum shape move must be included in the shape tables. For horizontal movement, a trailing byte #$00 may be needed, depending on how the shape is drawn. If the jump in position from one paddle read to the next is greater than the border in vertical movement, or larger than one byte in horizontal movement, then shape fragments will be left on the screen. Because the paddle routines we've used so far provide for virtually instantaneous movement, moving the paddle (or more easily the joystick) rapidly does produce large jumps. Try this with Program 6-1. Introduce a border of size 5 or so and convert to a DRAW-DRAW routine; then move the paddle slowly—okay. Then move it rapidly—interesting pattern, no? The solution to this problem (aside from huge, unworkable borders or movement limiters on your paddles) is to limit the maximum shape move regardless of paddle movement. This not only eliminates the DRAW-DRAW problem, but also provides for a smoother, more pleasing effect. In the next program (Program 15-2), we're going to modify Program 6-2 (horizontal movement of the man shape) by limiting the movement to a maximum of 5 bit positions at a time.

Program 15-2 is the same as Program 6-2 except for the PDLE subroutine, so we'll limit our discussion to that part of the program. Examining the flowchart will make this discussion easier to follow. The flowchart for Program 15-2 is on page 294. The two salient memory storage locations are MHORIZ, which contains the Y value used to calculate the shape position, and PDL, which contains the Y value from the most current paddle read.

If MHORIZ is larger than PDL, we want to subtract 5 from MHORIZ but not go below zero. After the subtraction, if MHORIZ is >= PDL, we continue with the program and use MHORIZ to calculate the new shape position. If MHORIZ < PDL, we don't want to go beyond the paddle position, so we set MHORIZ equal to PDL and then continue with the shape draw. If MHORIZ initially equals PDL, we set MHORIZ equal to PDL and continue. If MHORIZ is initially smaller than PDL, we add 5 to MHORIZ but only if it is below 250 so that we don't go beyond 255. After adding 5, if MHORIZ > PDL we set MHORIZ equal to PDL, again not to go beyond the paddle position. All this occurs just once each cycle, thus limiting the shape movement to a maximum of 5 bit positions in either direction.

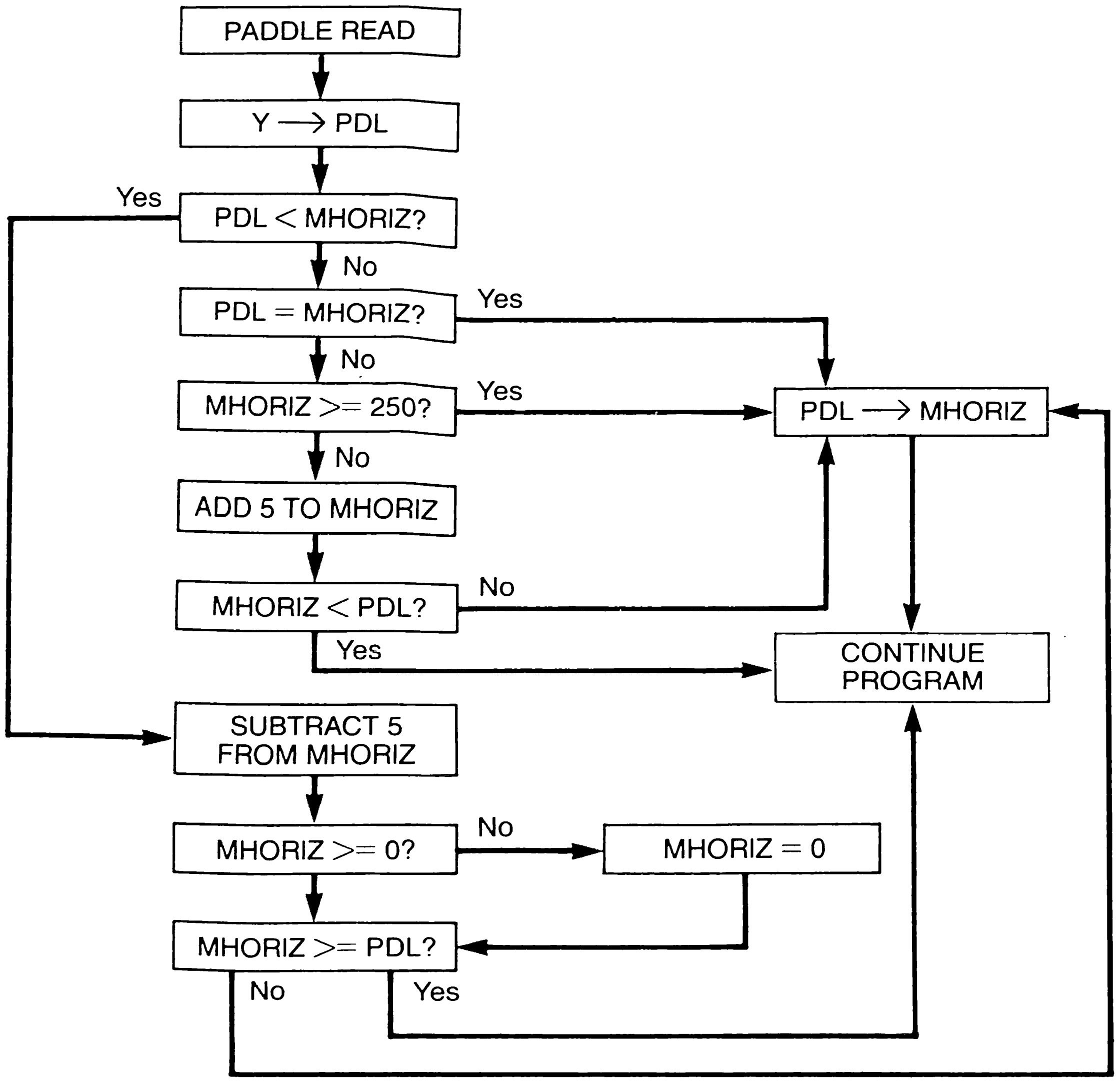
```
]PROGRAM 15-2
:ASM
                     *PADDLE OR JOYSTICK CONTROL OF HORIZONTAL MOVEMENT
                     *PADDLE SMOOTHING ROUTINE
                                ORG  $6000
6000: 4C 40 60                  JMP  PGM
                     LINE       DS   1
                     LINEA      DS   1
                     DEPTH      DS   1
                     HORIZ      DS   1
                     XCOUNT     DS   1
                     DELAY      DS   1
                     TEMP       DS   39
                     PDL        DS   1
                     MHORIZ     DS   1
                     GRAPHICS   =    $C050
                     MIXOFF     =    $C052
                     HIRES      =    $C057
                     PAGE1      =    $C054
                     HIGH       =    $1B
                     LOW        =    $1A
```

```
                    WAIT     =     $FCA8
                    PREAD    =     $FB1E
                    *LOAD SHAPE ADDRESSES INTO SHPADR, LOW BYTE FIRST
                    *CONTINUE FOR ALL 7 SHAPES
6032: 5A            SHPADR   DFB   #<SHAPE1
6033: 61                     DFB   #>SHAPE1
6034: 81                     DFB   #<SHAPE2
6035: 61                     DFB   #>SHAPE2
6036: A8                     DFB   #<SHAPE3
6037: 61                     DFB   #>SHAPE3
6038: CF                     DFB   #<SHAPE4
6039: 61                     DFB   #>SHAPE4
603A: F6                     DFB   #<SHAPE5
603B: 61                     DFB   #>SHAPE5
603C: 1D                     DFB   #<SHAPE6
603D: 62                     DFB   #>SHAPE6
603E: 44                     DFB   #<SHAPE7
603F: 62                     DFB   #>SHAPE7
6040: AD 50 C0      PGM      LDA   GRAPHICS     ;HIRES,P.1
6043: AD 52 C0               LDA   MIXOFF
6046: AD 57 C0               LDA   HIRES
6049: AD 54 C0               LDA   PAGE1
604C: A9 00                  LDA   #$00         ;CLEAR SCREEN 1
604E: 85 1A                  STA   LOW
6050: A9 20                  LDA   #$20
6052: 85 1B                  STA   HIGH
6054: A0 00         CLR1     LDY   #$00
6056: A9 00                  LDA   #$00
6058: 91 1A         CLR      STA   (LOW),Y
605A: C8                     INY
605B: D0 FB                  BNE   CLR
605D: E6 1B                  INC   HIGH
605F: A5 1B                  LDA   HIGH
6061: C9 40                  CMP   #$40
6063: 90 EF                  BLT   CLR1
6065: A9 60                  LDA   #$60         ;LOAD DELAY
6067: 8D 08 60               STA   DELAY
606A: A2 B7                  LDX   #$B7         ;DRAW LINE
606C: A0 00                  LDY   #$00
606E: BD 71 64               LDA   HI,X
6071: 85 1B                  STA   HIGH
6073: BD 31 65               LDA   LO,X
6076: 85 1A                  STA   LOW
6078: A9 7F                  LDA   #$7F
607A: 91 1A         LN       STA   (LOW),Y
607C: C8                     INY
607D: C0 27                  CPY   #$27
607F: 90 F9                  BLT   LN
                    ********** MAIN PROGRAM **********
6081: 20 96 60               JSR   INIT         ;SET LINE & DEPTH
6084: 20 A5 60      PADDLE   JSR   PDLE         ;READ PADDLE 0
6087: 20 10 61               JSR   DRAW         ;DRAW
608A: AD 08 60               LDA   DELAY
608D: 20 A8 FC               JSR   WAIT         ;DELAY
6090: 20 10 61               JSR   DRAW         ;ERASE
6093: 4C 84 60               JMP   PADDLE       ;READ PADDLE AGAIN
```

```
                  ********** SUBROUTINES **********
6096: A9 AA       INIT     LDA  #$AA
6098: 8D 03 60             STA  LINE
609B: 8D 04 60             STA  LINEA
609E: 18                   CLC
609F: 69 0D                ADC  #$0D
60A1: 8D 05 60             STA  DEPTH
60A4: 60                   RTS
                  **********************
60A5: A2 00       PDLE     LDX  #$00
60A7: 20 1E FB             JSR  PREAD        ;READ PADDLE 0
60AA: 8C 30 60             STY  PDL
60AD: 98                   TYA
60AE: CD 31 60             CMP  MHORIZ       ;IF MHORIZ > PDL,
60B1: 90 20                BLT  PADDLE3        SUBTRACT 5 FROM MHORIZ
60B3: CD 31 60             CMP  MHORIZ       ;IF MHORIZ = PDL,
60B6: F0 12                BEQ  PADDLE1        SET MHORIZ = PDL
60B8: AD 31 60             LDA  MHORIZ       ;IF MHORIZ < PDL,
60BB: C9 FA                CMP  #$FA           BUT >= 250,
60BD: B0 0B                BGE  PADDLE1          SET MHORIZ = PDL
60BF: AD 31 60             LDA  MHORIZ       ;IF < 250, ADD 5 TO MHORIZ
60C2: 18                   CLC
60C3: 69 05                ADC  #$05
60C5: CD 30 60             CMP  PDL          ;DON'T GO PAST PDL POSITION
60C8: 90 03                BLT  PADDLE2
60CA: AD 30 60    PADDLE1  LDA  PDL
60CD: 8D 31 60    PADDLE2  STA  MHORIZ
60D0: 4C EB 60             JMP  PADDLE5
60D3: AD 31 60    PADDLE3  LDA  MHORIZ       ;SUBTRACT 5 FROM MHORIZ
60D6: 38                   SEC
60D7: E9 05                SBC  #$05
60D9: B0 05                BGE  PADDLE4      ;BRANCH IF >= 0
60DB: A9 00                LDA  #$00         ;IF < 0,
60DD: 8D 31 60             STA  MHORIZ         SET MHORIZ = 0
60E0: CD 30 60    PADDLE4  CMP  PDL          ;DON'T GO PAST PDL POSITION
60E3: B0 E8                BGE  PADDLE2
60E5: AD 30 60             LDA  PDL
60E8: 4C CD 60             JMP  PADDLE2
60EB: AC 31 60    PADDLE5  LDY  MHORIZ
60EE: B9 6B 62             LDA  BYTETBL,Y    ;CONVERT TO SCREEN BYTE (0 - 36)
60F1: 8D 06 60             STA  HORIZ
60F4: B9 6E 63             LDA  OFFSET,Y     ;GET SHAPE NUMBER
60F7: 0A                   ASL               ;LOAD SHAPE INTO TEMP
60F8: AA                   TAX
60F9: BD 32 60             LDA  SHPADR,X
60FC: 85 1A                STA  LOW
60FE: BD 33 60             LDA  SHPADR+1,X
6101: 85 1B                STA  HIGH
6103: A0 00                LDY  #$00
6105: B1 1A       LOAD     LDA  (LOW),Y
6107: 99 09 60             STA  TEMP,Y
610A: C8                   INY
610B: C0 27                CPY  #$27
610D: 90 F6                BLT  LOAD
610F: 60                   RTS
```

```
                    131  ************************
6110: A9 00         132  DRAW     LDA  #$00
6112: 8D 07 60      133           STA  XCOUNT
6115: AE 03 60      134  DRAW1    LDX  LINE
6118: AC 06 60      135           LDY  HORIZ
611B: BD 71 64      136           LDA  HI,X
611E: 85 1B         137           STA  HIGH
6120: BD 31 65      138           LDA  LO,X
6123: 85 1A         139           STA  LOW
6125: AE 07 60      140           LDX  XCOUNT
6128: B1 1A         141           LDA  (LOW),Y
612A: 5D 09 60      142           EOR  TEMP,X
612D: 91 1A         143           STA  (LOW),Y
612F: C8            144           INY
6130: B1 1A         145           LDA  (LOW),Y
6132: 5D 0A 60      146           EOR  TEMP+1,X
6135: 91 1A         147           STA  (LOW),Y
6137: C8            148           INY
6138: B1 1A         149           LDA  (LOW),Y
613A: 5D 0B 60      150           EOR  TEMP+2,X
613D: 91 1A         151           STA  (LOW),Y
613F: EE 07 60      152           INC  XCOUNT
6142: EE 07 60      153           INC  XCOUNT
6145: EE 07 60      154           INC  XCOUNT
6148: EE 03 60      155           INC  LINE
614B: AD 03 60      156           LDA  LINE
614E: CD 05 60      157           CMP  DEPTH
6151: 90 C2         158           BLT  DRAW1
6153: AD 04 60      159           LDA  LINEA        ;RESET LINE
6156: 8D 03 60      160           STA  LINE
6159: 60            161           RTS
615A: 00 0E 01      162  SHAPE1   HEX  000E01000E01000E01    ;SHAPE TABLES
615D: 00 0E 01 00 0E 01
6163: 00 44 01      163           HEX  004401007F00601F00
6166: 00 7F 00 60 1F 00
616C: 30 1F 00      164           HEX  301F00181F00001F00
616F: 18 1F 00 00 1F 00
6175: 00 1F 00      165           HEX  001F00001B00403100
6178: 00 1B 00 40 31 00
617E: 60 60 00      166           HEX  606000
6181: 00 1C 02      167  SHAPE2   HEX  001C02001C02001C02
6184: 00 1C 02 00 1C 02
618A: 00 08 03      168           HEX  000803007E01003E00
618D: 00 7E 01 00 3E 00
6193: 00 3F 00      169           HEX  003F00403F00003E00
6196: 40 3F 00 00 3E 00
619C: 00 3E 00      170           HEX  003E00003600003600
619F: 00 36 00 00 36 00
61A5: 00 63 00      171           HEX  006300
61A8: 00 38 04      172  SHAPE3   HEX  003804003804003804
61AB: 00 38 04 00 38 04
61B1: 00 10 06      173           HEX  001006007C03007C00
61B4: 00 7C 03 00 7C 00
61BA: 00 7C 00      174           HEX  007C00007E00007C00
61BD: 00 7E 00 00 7C 00
```

```
61C3: 00 38 00   175          HEX  003800003800006C00
61C6: 00 38 00 00 6C 00
61CC: 00 46 01   176          HEX  004601
61CF: 00 70 08   177  SHAPE4  HEX  007008007008007008
61D2: 00 70 08 00 70 08
61D8: 00 20 0C   178          HEX  00200C007807007801
61DB: 00 78 07 00 78 01
61E1: 00 78 01   179          HEX  007801007801007801
61E4: 00 78 01 00 78 01
61EA: 00 70 00   180          HEX  007000007000007000
61ED: 00 70 00 00 70 00
61F3: 00 70 00   181          HEX  007000
61F6: 00 60 11   182  SHAPE5  HEX  006011006011006011
61F9: 00 60 11 00 60 11
61FF: 00 40 18   183          HEX  00401800700F007003
6202: 00 70 0F 00 70 03
6208: 00 70 03   184          HEX  007003007803007003
620B: 00 78 03 00 70 03
6211: 00 60 01   185          HEX  006001006001003003
6214: 00 60 01 00 30 03
621A: 00 18 06   186          HEX  001806
621D: 00 40 23   187  SHAPE6  HEX  004023004023004023
6220: 00 40 23 00 40 23
6226: 00 00 31   188          HEX  00003100601F006007
6229: 00 60 1F 00 60 07
622F: 00 70 07   189          HEX  007007007807006007
6232: 00 78 07 00 60 07
6238: 00 60 07   190          HEX  006007006006006006
623B: 00 60 06 00 60 06
6241: 00 30 0C   191          HEX  00300C
6244: 00 00 47   192  SHAPE7  HEX  000047000047000047
6247: 00 00 47 00 00 47
624D: 00 00 62   193          HEX  00006200403F00700F
6250: 00 40 3F 00 70 0F
6256: 00 58 0F   194          HEX  00580F004C0F00400F
6259: 00 4C 0F 00 40 0F
625F: 00 40 0F   195          HEX  00400F00400D006018
6262: 00 40 0D 00 60 18
6268: 00 30 30   196          HEX  003030
                      BYTETBL
                      OFFSET
                      HI
                      LO

1521 bytes

Symbol table - numerical order:

  LOW      =$1A     HIGH     =$1B     LINE     =$6003   LINEA    =$6004
  DEPTH    =$6005   HORIZ    =$6006   XCOUNT   =$6007   DELAY    =$6008
  TEMP     =$6009   PDL      =$6030   MHORIZ   =$6031   SHPADR   =$6032
  PGM      =$6040   CLR1     =$6054   CLR      =$6058   LN       =$607A
  PADDLE   =$6084   INIT     =$6096   PDLE     =$60A5   PADDLE1  =$60CA
  PADDLE2  =$60CD   PADDLE3  =$60D3   PADDLE4  =$60E0   PADDLE5  =$60EB
  LOAD     =$6105   DRAW     =$6110   DRAW1    =$6115   SHAPE1   =$615A
  SHAPE2   =$6181   SHAPE3   =$61A8   SHAPE4   =$61CF   SHAPE5   =$61F6
  SHAPE6   =$621D   SHAPE7   =$6244   BYTETBL  =$626B   OFFSET   =$636E
  HI       =$6471   LO       =$6531   GRAPHICS=$C050    MIXOFF   =$C052
  PAGE1    =$C054   HIRES    =$C057   PREAD    =$FB1E   WAIT     =$FCA8
```



Run Program 15-2 and compare it to Program 6-2. I think you'll agree the effect is more pleasing and is reminiscent of the type of paddle control one sees in Invader-type games.

The paddle smoothing routine can be used in any program using paddles to move shapes in any direction and the maximum speed of movement can be altered simply by changing the value to be added or subtracted. This routine also allows one to incorporate a DRAW-DRAW routine into the program. For vertical animation, we simply limit the maximum move to the border size. For horizontal animation, we need only limit the maximum move to one byte or less.

One final note. I haven't combined the paddle smoothing routine with the paddle movement test of Program 15-1. This is something for you to do, as we'll talk about in the last chapter.

16

Integrating BASIC with Assembly Language Programs

There once was a woman named Kit,
Whose husband gave her a fit.
Computing all night
He neglected her plight
So they drifted apart, bit by bit.

(This has nothing to do with this chapter, but it's hard writing these things.)

There isn't anything that can be done in BASIC that can't be done in assembly language, but for some things BASIC is much easier. Complex arithmetic, for example, is much simpler using BASIC. In assembly language, you can add, subtract, multiply, and divide, but in BASIC a whole host of arithmetic functions are available, such as SQR, ABS, INT, SIN, COS, TAN, RND, EXP, LOG, etc., and working with formulas is made simpler with the DEF FN instruction (see the Apple BASIC manual for details). Of course, all these functions can be derived from the four basics of adding, subtracting, multiplying, and dividing, but if speed is not required, it's much easier to let the BASIC interpreter do it for you. If speed is required (let's say you want to plot sine curves on the hi-res screen—easy but slow in BASIC), you'll need to use assembly language. Deriving sine curve equations from the four basic arithmetic functions is about as much fun as defleaing your dog but fortunately, if you need to do this, there are texts on the market that deal with complex number manipulations using assembly language.

Printing to the text screen is often used with hi-res graphics programs, either for displaying whole page instructions or for printing on the bottom four lines of page 1 with the mixed text and graphics mode. (This is different from printing on the hi-res screen itself—here one needs to use shapes in the form of letters in the same way as we used number shapes for displaying the score in the game program.) Text printing can be done using BASIC or assembly language and in fact is relatively easy with assembly language, because one can use several built-in Apple subroutines to take care of the housekeeping chores. But, to my mind, nothing is easier than the BASIC PRINT statement. The only advantage of assembly language for text printing is speed, but this is like saying you can travel a distance of 1 foot faster going 100 mph than going 50 mph. Printing in BASIC is so fast, unless you're using some kind of convoluted code, that any speed advantage of assembly language is more academic than real.

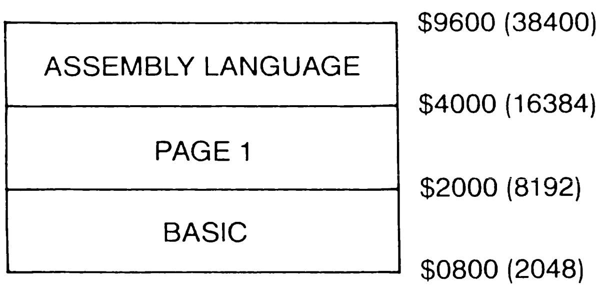
MEMORY ALLOCATION

Dealing with a program that uses both BASIC and assembly language, or with a program that uses both BASIC and hi-res graphics whether or not assembly language is also used, requires that attention be paid to how memory is allocated. We have to be careful that BASIC, assembly language, and the hi-res screens don't run into each other, that is, do not occupy the same memory locations; otherwise, we will be left with an unworkable program. To see how to do this, let's consider first how BASIC uses memory.

On startup, the Apple assigns $800 (2048) to the bottom of BASIC and $9600 (38400) to the top (for machines with 48K minimum RAM,– see Chapter 2). The reason a top has to be assigned is that while the BASIC program starts at the bottom, variables are stored after the end of the last program line, and string variables are stored at the top and work their way down. Thus, any non-BASIC program code such as assembly language or a hi-res screen between the top and bottom may interfere with the BASIC program itself, especially if string variables are used. Even if they're not, it's always a good idea to reserve an area of memory for BASIC to ensure no overlap with the hi-res screens or with assembly language code. There are two solutions to this problem—we either change the top or change the bottom of BASIC, the particular choice depending on how much memory we want to allocate to the different parts of the program and which hi-res screen we want to use.

There are four basic situations to consider:

1. Page 1 hi-res screen, BASIC below Page 1, assembly language above Page 1:



ASSEMBLY LANGUAGE 22000 BYTES
BASIC 6000 BYTES

The bottom of BASIC is the startup address, $800 (2048). We want to move the top, the area for string variable storage, to the bottom of hi-res screen Page 1, which starts at $2000 (8192). We do this in the beginning of the BASIC program by

```
1 HIMEM:8192
```

(Note that BASIC uses only decimal addresses, not hex.) This instruction ensures that BASIC will occupy a memory block that will not be interfered with by the hi-res screen or the assembly language code. The assembly language program itself could be given a starting address of $4000, that is, just above Page 1. The result of all this is that BASIC would have about 6000 bytes of available memory, whereas the assembly language program would have about 22,000 bytes. (If we want to use both hi-res screens, the assembly language program would be started at $6000 and would have about 14,000 bytes of memory.)

2. Page 2 hi-res screen. BASIC below Page 2, assembly language above Page 2:

	$9600 (38400)
ASSEMBLY LANGUAGE	
	$6000 (24576)
PAGE 2	
	$4000 (16384)
BASIC	
	$0800 (2048)

ASSEMBLY LANGUAGE 14000 BYTES
BASIC 14000 BYTES

Here we want to move the top of BASIC to $4000 and we do this by

```
1 HIMEM:16384
```

The assembly language code would start at $6000 and have about 14,000 bytes of memory, and BASIC would have also about 14,000.

3. Page 1 hi-res screen, BASIC above Page 1, assembly language below Page 1:

	$9600 (38400)
BASIC	
	$4000 (16384)
PAGE 1	
	$2000 (8192)
ASSEMBLY LANGUAGE	
	$0800 (2048)

ASSEMBLY LANGUAGE 6000 BYTES
BASIC 22000 BYTES

Moving the bottom of BASIC is a little more complicated than moving the top. There is no single command to do this; rather a series of POKEs is required. Locations 103 and 104 have to be POKEed with certain values and the new start of BASIC has to be POKEed with zero because BASIC must always start with zero in the first position. There is a formula that can be used to calulate the values to be POKEed into 103 and 104, but the easiest thing to do is incorporate the formula into a BASIC instruction itself and let the program do the calculating for you. What we do is set up a separate program called a "loader" program and use it both to change the bottom of BASIC and run the main program. The "loader" program consists of one line (make sure you save the program before running it because it self-destructs on running):

```
1 LOC = 16384 + 1:POKELOC - 1,0:POKE 103,LOC - INT(LOC/256) * 256:
  POKE 104,INT (LOC/256): PRINT CHR$(4); "RUN PROGRAM"
```

Running this program will set the bottom of BASIC to $4000 and will run the main program labeled PROGRAM, assuming of course it's on the same disk. In this case, BASIC will have about 22,000 bytes of available memory and the assembly language program about 6,000 bytes, assuming we start it at $800.

4. Page 2 hi-res screen, BASIC above page 2, assembly language below Page 2:

	$9600 (38400)
BASIC	
	$6000 (24576)
PAGE 2	
	$4000 (16384)
ASSEMBLY LANGUAGE	
	$0800 (2048)

```
ASSEMBLY LANGUAGE  14000 BYTES
BASIC              14000 BYTES
```

The only change here is defining LOC in the "loader" program as 24576 + 1. Now both BASIC and the assembly language program will have about 14,000 bytes of available memory.

These are the four basic memory allocation situations, but variations are sometimes required. For example, and as mentioned above, if we want to use both hi-res screens, things would have to be shifted around, removing memory from either BASIC or the assembly language program, depending on the particular configuration we want. Also, because we can start the assembly language program anywhere, the actual memory available for assembly language is variable.

Other situations may require other changes. For example, suppose we're using the number 3 configuration with BASIC above Page 1. If our assembly language program requires 8,000 bytes instead of the 6,000 available, we could move the bottom of BASIC to around 19,000 instead of 16,384 and use the extra memory for assembly language code. We could use this memory block, for example, to store our line address and offset tables. Assembly language programs do not require a continuous block of uninterrupted memory, but when we split up such a program, we have to be careful where we do it. For example, we wouldn't want an interruption in the middle of a draw routine. We can, however, place any block of code that is accessed only by its label anywhere we want. The only caveat, as discussed before, is that relative branch instructions have a range limited to 127 bytes forward and 128 bytes back—in these cases, we use the relative branch to go to a nearby JMP instruction, which has no range limitation. In the example cited above, we would set up our main program with ORG $800 and place the line address and offset tables starting at $4000, making sure they do not extend into the start of BASIC.

The particular configuration we would choose obviously depends on the program requirements. If we need Page 1 to display mixed text and graphics, we must use configuration 1 or 3. If the assembly language program is long and BASIC short, we would choose 1; conversely, configuration 3 would be appropriate for long BASIC and short assembly language programs. Keep in mind that because we can place assembly language blocks almost anywhere we want and we can vary the top and bottom of BASIC, there is a large amount of flexibility in how to allocate available memory for any particular program application.

If your program lengths are running close to the available memory limits, it's important to know the program size so you can plan accordingly. This is no problem with assembly language programs, as most assemblers will display the length in bytes after assembly. To determine the length of a BASIC program in bytes, load the program and type in the following instructions (with thanks to Beagle Bros.):

```
PRINT (PEEK (175) ) + PEEK (176) * 256) - (PEEK (103) + PEEK (104) * 256)
```

If you find your programs are too large to fit, don't despair just yet. Assembly language programs can be shortened by writing more efficient code, but this is probably applicable only to very experienced programmers. An easy trick to extend total available memory is to include the following as the first line of your BASIC program:

```
1 PRINT CHR$(4);"MAXFILES1"
```

This extends the upper limit of memory from $9600 (38400) to $9AA6 (39590) (for 48K minimum machines), making available an extra 1,190 bytes. The price you pay for this is that only one text file can be open at one time (see the Apple DOS Programmer's Manual for more details). Another thing you can do is use one of the commercially available optimizing programs, such as those available from Beagle Bros., to crunch your BASIC programs. You'd be amazed at how much space can be saved using one of these utilities, but save this for last—such crunched programs are virtually impossible to edit. If you're really desperate, try one of the DOS mover programs (if you have 64K)—you can realize some 10,000 bytes of extra memory this way.

ZERO PAGE USAGE

We mentioned in a previous chapter that we have to be careful in choosing which zero page addresses to use in our assembly language programs. This is because zero page is used extensively by Applesoft BASIC and DOS and so if we're using BASIC along with our assembly language program or if we're using DOS commands, either from BASIC or from assembly language, we have to search for "open" zero page addresses and there aren't many. For the Apple IIe, safe zero page addresses are $06 to $09, $1A to $1C, $EB to $EF, and $F9 to $FC. These are probably okay for other Apple IIs but I would check the Reference Manual for your particular machine just to make sure. Of course if you're not using BASIC or DOS, then any zero page address can be used, but it's probably best to stick with the safe ones—you never know when you might be adding DOS commands or BASIC to your assembly language program.

GRAPHICS AND TEXT COMMANDS FROM BASIC

You've seen some of these instructions before in Chapter 3 and they are all described in the Apple BASIC manual but some rather obtusely—a brief review is worthwhile.

GR Clears and displays low resolution screen.

HGR Clears and displays hi-res screen Page 1 (mixed text and graphics with the bottom four lines displaying text).

HGR2 Clears and displays hi-res screen Page 2.

TEXT Displays the full screen text page without clearing it.

HOME Clears but does not display the text page and sends the cursor to the top left position. When used with hi-res screen Page 1 in mixed text and graphics mode, the bottom four lines are cleared and the cursor is positioned at VTAB 21 without affecting the graphics display. The combined instructions TEXT:HOME will display and clear the entire text page regardless of which hi-res screen is being used.

POKE 49232,0 (or POKE –16304,0) (In this and the following examples, either number can be POKEed but Integer BASIC requires poking the negative number.) Accesses the graphics mode, either lo-res or hi-res depending on the status of other soft switches, without clearing the graphics screen.

POKE 49233,0 (POKE –16303,0) Selects the text page without clearing it: the text page margins can be altered to produce a text "window;" unlike the TEXT command, this instruction leaves the text "window" settings intact.

POKE 49234,0 (POKE –16302,0) Selects full screen graphics for hi-res screen Page 1.

POKE 49235,0 (POKE –16301,0) Selects mixed text and graphics for hi-res screen Page 1 (not necessary after an HGR if full screen graphics has not been selected).

POKE 49236,0 (POKE –16300,0) Selects Page 2 without clearing it.

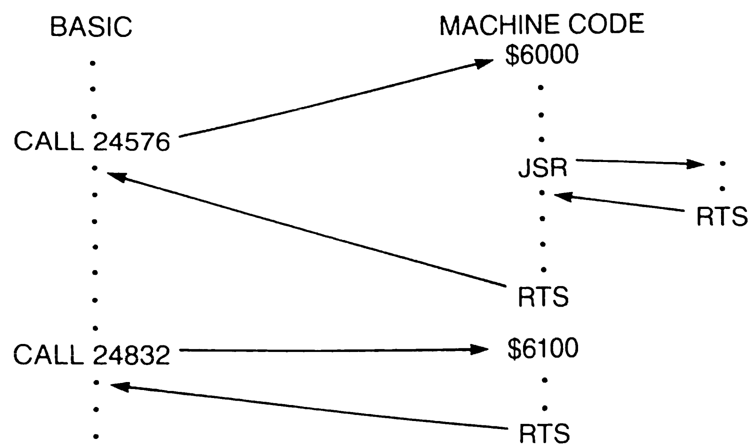
POKE 49237,0 (POKE –16299,0) Selects Page 1 without clearing it.

POKE 49238,0 (POKE –16298,0) Selects low resolution mode (not necessary after a GR).

POKE 49239,0 (POKE –16297,0) Selects high resolution mode (not necessary after an HGR or HGR2).

ACCESSING ASSEMBLY LANGUAGE PROGRAMS FROM BASIC

When combining BASIC with assembly language, program control essentially rests with the BASIC program. A particular assembly language program address is accessed from BASIC by the instruction CALL *address* (decimal). Thus, the command CALL 24576 would send the program to the machine code beginning at $6000. The program returns to BASIC when it reaches an RTS opcode that does not follow a JSR. If there is no such RTS, the program remains in the assembly language portion. For example:



To see how this works in an actual program, let's use BASIC in our game program to display the game instructions before starting. On our disk we would have the game program labeled GAME. The BASIC program would look like this:

```
10 PRINT CHR$(4);"BLOAD GAME,A$6000"
20 TEXT:HOME
30 PRINT "GAME INSTRUCTIONS ........ PRESS ANY KEY TO CONTINUE"
40 GET A$
50 HGR:POKE -16302,0
60 CALL 24576
```

One of the advantages of using BASIC is that BASIC commands often can substitute for assembly language code. In this instance, for example, HGR is used to display and clear the Page 1 hi-res screen and POKE −16302,0 displays full screen graphics. As this is done in the beginning, speed is not required, and the call to Page 1 full screen graphics and the clear screen routines in the GAME program become unnecessary and can be deleted. Once the call to $6000 is made, the program stays entirely in the GAME program, because there is no RTS not preceded by a JSR.

Let's modify the GAME even further by changing the restart protocol. In the STOP2 subroutine, instead of the *press any key* routine we substitute an RTS. When the game ends, this will send the program back to BASIC where we will now display another text screen containing, for example, a scoring summary:

```
70 TEXT:HOME
80 PRINT "GAME SUMMARY. . . . PRESS ANY KEY TO CONTINUE"
90 GET A$
100 GOTO 50
```

The TEXT instruction calls the text screen and HOME clears it. We restart the game by going back to line 50. Alternatively, if we want to display the game instructions again, line 100 would read GOTO 20. Other variations are possible.

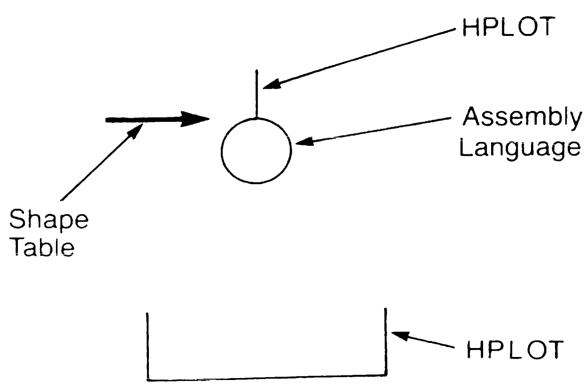
Suppose we want to restart just by going back to the game instructions:

```
70 GOTO 20
```

This combination of printing in BASIC and displaying graphics in assembly language is quite easy and very useful, not only for game programs but also for a multitude of other applications. In addition, BASIC can be used not only for printing but also for graphics itself, in conjunction with assembly language graphics. This is possible because the hi-res screen doesn't care where its instructions come from and so one can freely intermix graphics from BASIC, assembly language, and even from Apple shape tables. The only caveat is that pleasant results are obtained only if the non-assembly language graphics do not involve themselves in routines that require speed and smooth animation. Perhaps the greatest utility of this type of intermixing is in educational programs. Such programs generally do not involve continuous, rapid animation as in games, but rather present a series of lessons, each one consisting of some text and a graphics presentation that consists only partly of animation routines.

To see how we can profitably mix text and graphics from a variety of sources, let's design a small educational program that illustrates the principle that objects fall down unless restrained (the profit comes from selling the program to kindergarten computer workshops). We're going to use hi-res screen Page 1 with mixed text and graphics and use the bottom four lines for the explanatory text. Because our BASIC program is small, we'll place it below Page 1 with HIMEM:8192. We're also going to use a shape table and we'll load this above Page 1 at \$4000 (16384). The assembly language program will be loaded above the shape table, at \$6000 (24576).

The screen will show a ball suspended by a rope above a container.



After some text instructions, the rope will be cut at a site indicated by an arrow—the arrow will then disappear and the ball will fall into the container. The container and rope will be drawn from BASIC by HPLOTting. The arrow will come from the shape table (designed with the aid of a utility program such as Apple Mechanic from Beagle Bros.) and the ball will be drawn and animated with our assembly language program. The Page 1 screen will be displayed and cleared from BASIC and we can begin the assembly language program at \$6000 with just drawing the ball. This draw routine ends with an RTS, say at \$64FF, and thus will return to BASIC once the ball is drawn. At \$6500 (25856), the program continues with the ball animation and also ends with an RTS to get back to

BASIC again. The program would look like this (the shape table and assembly language program will already have been BSAVEd on the disk:

```
1 HIMEM:8192      :REM BASIC BELOW PAGE 1
10 PRINT CHR$(4);"BLOAD SHAPE TABLE,A$4000"
20 PRINT CHR$(4);"BLOAD ASSEMBLY LANGUAGE,A$6000"
30 SH = 16384     :REM LOCATION OF SHAPE TABLE
40 POKE 232,SH - INT(SH/256) * 256:POKE 233, INT(SH/256) :REM TELLS PROGRAM
   WHERE SHAPE TABLE IS LOCATED
50 HGR   :REM SELECTS HI-RES PAGE 1
60 HCOLOR = 3   :REM COLOR WHITE
70 ROT = 0:SCALE = 1 :REM NEEDED FOR SHAPE TABLE DRAW
80 HOME: VTAB21:PRINT "WHAT WILL HAPPEN WHEN THE ROPE IS CUT?":PRINT "PRESS ANY
   KEY TO CONTINUE" :REM PRINTS ON BOTTOM 4 LINES
90 HPLOT 100,50 TO 100,100 TO 150,100 TO 150,50 :REM PLOTS CONTAINER
100 HPLOT 125,10 TO 125,20 :REM PLOTS ROPE
110 DRAW 1 AT 125,20 :REM DRAWS ARROW
120 CALL 24576 :REM DRAWS BALL
130 GET A$ :REM WAIT FOR KEYPRESS
140 HOME: VTAB21:PRINT "LET'S DO IT. PRESS ANY KEY TO CUT THE ROPE" :REM CHANGES
    TEXT IN BOTTOM 4 LINES BUT LEAVES GRAPHICS INTACT
150 GET A$
160 XDRAW 1 AT 125,20 :REM ERASES ARROW
170 CALL 25856 :REM MOVES BALL DOWN
180 HOME: VTAB21:PRINT "YOU WERE RIGHT! THE BALL FALLS":PRINT "PRESS ANY KEY TO
    CONTINUE
190 GET A$
200 TEXT:HOME: PRINT "IF YOU LIKED THIS PROGRAM, TELL MOMMY TO BUY IT." :REM
    PRINTS ON TEXT PAGE
```

The variations on this theme are endless. We could clear the screen with HGR and continue with more graphics from any source; we could draw the container, rope, and arrow in color by specifying a color with HCOLOR and, of course, draw the ball in color in the assembly language program; we could make a larger container by changing SCALE; we could switch back and forth from text to graphic screens without erasing them by using the appropriate POKEs, and so on. The reason this works is that HPLOT and DRAW are very fast for simple shapes that are displayed and not moved—assembly language is required only for the animation. And let me emphasize that assembly language is indeed required—moving the ball around from BASIC or shape tables would produce an animation that would immediately mark you as a rank amateur, deserving only of scorn. From personal experience, I can tell you that professional-looking animation is a strong selling point for these types of programs.

17

Suggestions for Game Modification

We've covered quite a bit since constructing the game program, but, of course, the coverage has not been exhaustive (and I don't mean it hasn't been tiring). My hope is that this book has provided the necessary background in hi-res assembly language graphics so that you can now profitably examine more advanced texts and those rather obtuse (I hope now less obtuse) magazine articles that pop up now and then to learn even more about this subject, either for constructing your own games or indeed for any of the many other applications that find hi-res graphics useful. You may not know enough yet to construct a really super arcade-type game, but you certainly have the knowledge to produce professional results for interesting games and for educational and other types of programs. And, of course, you've also learned something about how to use assembly language other than just to move numbers around.

As mentioned in the Introduction, any learning process is enhanced by doing and not just observing. In this spirit, below I will discuss suggestions for modifying the game program using techniques covered in Parts One and Two that were not incorporated into the game. You may find that going through the exercises in this chapter will teach you more about hi-res graphics and assembly language than all the other chapters combined, and so I urge you to sharpen your typing fingers and go to work.

1. Use the DRAW-DRAW protocol for both the man and the plane, making sure to incorporate the appropriate borders. As neither shape is used for collision detection, DRAW-DRAW will work and will produce smoother animation. For the man shape, use the paddle smoothing routine (necessary for DRAW-DRAW as discussed in Chapter 15) and the test for non-movement of paddle to eliminate flicker.

2. Work on the sound routines for the plane and explosions and perhaps for the bullet firing.

3. Have planes appear at several different line positions and have some going right to left instead of just left to right. Make sure each line position is some multiple of 8 from the bullet firing line to ensure collision detection with the bullet; alternatively, use multiple line collision testing with the bullet shape.

4. Have the planes drop bombs and use the bomb shape for collision testing. Use the line position of a collision to determine what the bomb has hit—if at the bottom, it hits the bottom line; above the bottom, but not above the top of the man, it hits the man; above the man, it hits a bullet. Include an explosion routine for each collision. Have the game end if the man is hit.

5. Change the scoring protocol to decrement by 1 each time a bullet is fired, increment by 3 for each plane hit and by 5 for each bomb hit. A plane is hit if only a bullet detects a collision. A bomb is hit if both the bomb and bullet detect a collision.

6. Use BASIC to display the game instructions.

7. Draw the plane in color, changing the bullet shape to a width of 2 bits to ensure collision detection. Draw the explosion shapes in orange and yellow. Draw flickering orange lines directly behind the plane to simulate engine exhaust. Enlarge the man shape and draw in color.

8. Reprogram the game in double hi-res and in double hi-res color. For the latter, use dummy white shape tables for collision detection.

Well, that's it. Good luck—and remember, #$2B or not #$2B is not the only question.

Appendix: Assembly Language Commands

Not all assembly language commands for the Apple II 6502 microprocessor are listed here, mainly just those referenced in the text. In addition, the descriptions are not comprehensive. A complete set, with complete descriptions, can be found in texts on assembly language programming. I especially recommend *Assembly Lines: The Book,* by Roger Wagner, Roger Wagner Publishing Co., Santee, CA, 1982.

ADC (ADd with Carry) Adds the contents of a memory location or a direct value to the contents of the Accumulator, plus the Carry bit if it was set. The result is stored in the Accumulator. ADC is usually preceded by a CLC in case the Carry bit has inadvertently been set. A common use of ADC is to add two numbers together.

Example

```
CLC
LDA #$01
ADC #$01        ;Accumulator now contains #$02

CLC
LDA #$01
ADC $4000       ;$4000 contains #$04
STA $5000       ;$5000 now contains #$05
```

AND (Logical AND) Compares each bit of the Accumulator with each corresponding bit of the contents of a memory location or a direct value. If both bits are 1, the result is 1; otherwise the result is 0. The result is stored in the Accumulator. This command is useful for collision detections.

Example

Accumulator	0 0 0 1 1 1 0 1
Number	0 1 1 1 0 1 0 0
Result	0 0 0 1 0 1 0 0

ASL (Arithmetic Shift Left) Each bit of the Accumulator or the contents of a memory location is moved one position to the left. A 0 is placed in bit 0 (the rightmost bit) and the high bit is placed in the Carry. One use of ASL is to multiply by factors of two.

C <--- 7 6 5 4 3 2 1 0 <--- "0"

BCC (Branch on Carry Clear) The branch is taken if the Carry bit is clear; i.e., zero. The pseudo-op BLT (Branch if Less Than) can be used by some assemblers, because BCC is often used after a comparison instruction to test if the Accumulator holds a value less than a specified value; if it does, the Carry bit is clear and the branch is executed.

Example

```
LDA  #$05
CMP  #$06
BCC  CONTINUE    ;The branch is taken
```

BCS (Branch on Carry Set) The branch is taken if the Carry bit is set, i.e., 1.

The pseudo-op BGE (Branch if Greater or Equal) This can be used by some assemblers as BCS is often used after a comparison instruction to test if the Accumulator holds a value equal to or greater than a specified value; if it does, the Carry bit is set and the branch is executed.

Example

```
LDA  #$05
CMP  #$04
BCS  CONTINUE    ; The branch is taken
```

BEQ (Branch if EQual) Branches if the result of a previous operation is zero. It is often used to compare the value in the Accumulator or the contents of a memory location to a specified value, which itself can be the contents of a memory location or a direct value. If the values are equal, the branch is taken.

Example

```
LDA  LINE
CMP  DEPTH
BEQ  CONTINUE    ;The branch is taken if LINE = DEPTH
```

BEQ can also be used to test for a register reaching a zero value.

Example

```
        LDY #$06
LOOP    LDA $4000
        STA $5000
        DEY
        BEQ CONTINUE    ;The branch is taken when Y = 0
        JMP LOOP
```

BIT Compare Accumulator BITs with contents of memory. BIT can be used to access a soft switch without changing the contents of the Accumulator.

Example

```
BIT $C030    ;Tweaks speaker
```

BMI (Branch on MInus) Branches if any operation produces a result in the range #$80 to #$FF, i.e., high bit set. One use is to test for a keypress.

Example

```
LOOP    LDA $C000       ;If no key pressed, value < #$80
        BMI CONTINUE    ;Branches if key pressed
        JMP LOOP
```


BMI can also be used to terminate a loop when a value reaches any number from #$80 to #$FF.

Example

```
        LDY #$70
LOOP    DEY
        BMI CONTINUE    ;Branch taken when Y wraps around to #$FF
        JMP LOOP
```

BNE (Branch on Not Equal) Branches if the result of any operation is non-zero.

Example

```
LDA #$06
CMP #$05
BNE CONTINUE    ;Branch is taken
```

BNE can also be used in loops to test for non-zero.

Example

```
        LDY #$06
LOOP    DEY
        BNE LOOP        ;Branches until Y = 0
        RTS
```

BPL (Branch on PLus) Branches if any operation produces a result in the range #$00 to #$7F, i.e., high bit not set. BPL can be used to test for a key press.

Example

```
LOOP    LDA $C000       ;If no key pressed, value < #$80
        BPL LOOP        ;Branches until key is pressed
        JMP CONTINUE
```

BPL can also be used to terminate a loop when a value reaches any number outside the range #$00 to #$7F.

Example

```
        LDY #$70
LOOP    DEY
        BPL LOOP     ;Branches until Y = #$FF
        RTS
```

Note: Conditional branches are limited to 127 bytes forward and 128 bytes back.

BRK (BReaK) Halts execution of the program. This command is useful for debugging programs. By placing BRK at strategic locations, the program can be stopped and the status of the registers and memory locations examined.

CLC (CLear Carry) Clears the Carry bit; usually used preceding an ADC instruction in case the Carry bit has been set accidentally somewhere else in the program. It can also be used to force a branch.

Example

```
CLC
BCC CONTINUE    ;Branch always taken
```

CMP (CoMPare to Accumulator) Compares the value in the Accumulator to a direct value or to the contents of a memory location. CMP is used with comparison instructions for conditional branches.

Example

```
LDA #$06
CMP #$07
BCC CONTINUE
```

CPX (ComPare to X register) Compares the contents of the X register to a direct value or to the contents of a memory location; used with conditional branch instructions.

Example

```
        LDX #$00
LOOP    LDA LINE,X
        STA LINEA
        INX
        CPX #$05
        BCC LOOP        ;Branches until X = 5
        RTS
```

CPY (ComPare to Y register) Compares the contents of the Y register to a direct value or the contents of a memory location; see CPX.

DEC (DECrement) Decrements the contents of a memory location by one. If the location contains #$00, the value will wrap around to #$FF.

Example

```
LDA #$00
STA $4000
DEC $4000       ;$4000 now contains #$FF
```


DEX (DEcrement to X register) Decrements the X register by one; see DEC.

DEY (DEcrement the Y register) Decrements the Y register by one; see DEC.

EOR (Exclusive-OR with Accumulator) Each bit of the Accumulator is compared to the corresponding bit of a direct value or the contents of a memory location. If either bit is 1, the result is 1; if both bits are 1 or 0, the result is 0. The result is stored in the Accumulator. EOR is useful in drawing routines for both drawing and erasing.

Example

```
Accumulator     1 0 0 1 1 0 0 0
Number          0 1 0 1 0 1 1 0
                ---------------
Result          1 1 0 0 1 1 1 0
```

INC (INCrement memory) Increments the contents of a memory location by one. If the location contains #$FF, the value will wrap around to #$00 (see DEC).

INX (INcrement the X register) Increments the X register by one; see INC.

INY (INcrement the Y register) Increments the Y register by one; see INC.

JMP (JuMP to address) Sends the program to the specified address.

JSR (Jump to SubRoutine) Analogous to a GOSUB in BASIC, JSR sends the program to a subroutine at a specified address. When an RTS in the subroutine is encountered, the program returns to the program line immediately following the JSR (see RTS).

LDA (LoaD the Accumulator) Loads the Accumulator with a direct value or the contents of a memory location.

Example

```
LDA #$05        ;Accumulator contains #$05
LDA $4000       ;Accumulator contains contents of $4000
```

LDX (LoaD the X register) Loads the X register with a direct value or the contents of a memory location; see LDA.

LDY (LoaD the Y register) Loads the Y register with a direct value or the contents of a memory location; see LDX.

LSR (Logical Shift Right) Opposite of ASL; each bit of the Accumulator or the contents of a memory location is moved 1 position to the right. A zero is placed in the high bit and bit 0 (the rightmost bit) is placed in the Carry bit. One use of LSR is to divide by factors of 2.

```
"0" ---> 7 6 5 4 3 2 1 0 ---> C
```

NOP (No OPeration) This does what it says; no operation is performed, but time is used. NOP is used for debugging by disabling certain steps and can also be used as a time delay.

ORA (Inclusive OR with Accumulator) Compares each bit of the Accumulator with the corresponding bit of a direct value or the contents of a memory location. If either or both bits are 1, the result is 1; if both bits are 0, the result is 0. The result is stored in the Accumulator.

Example

Accumulator	0 0 1 1 0 0 1 1
Number	0 1 0 1 0 1 0 1
Result	0 1 1 1 0 1 1 1

ROL (ROtate Left) Each bit of the Accumulator or the contents of a memory location is moved one position to the left. The Carry bit is placed into bit 0 and is replaced by the high bit (see ASL).

```
C <--- 7 6 5 4 3 2 1 0 <--- C
```

ROR (ROtate Right) Each bit of the Accumulator or the contents of a memory location is moved one position to the right. The Carry bit is placed in the high bit and replaced by bit 0 (see ROL).

```
C ---> 7 6 5 4 3 2 1 0 ---> C
```

RTS (ReTurn from Subroutine) Returns the program to the line immediately following the JSR call to the subroutine (see JSR). An RTS without a preceding JSR is used to return the program to BASIC when the assembly language program is called from BASIC.

SBC (SuBtract with Carry) Subtracts the contents of a memory location or a direct value from the Accumulator and also subtracts the opposite of the

Carry. The result is stored in the Accumulator. SBC should always be preceded by SEC prior to the first subtraction.

Example

```
LDA #$05
SEC
SBC #$03          ;Accumulator now contains #$02
```

SEC (SEt Carry) Sets the Carry bit to 1. Used before a SBC instruction and also can be used to force a branch.

Example

```
SEC
BCS CONTINUE          ;Branch always taken
```

STA (STore Accumulator) Sends the contents of the Accumulator to a specified memory location. The Accumulator is not affected.

Example

```
STA $4000         ;$4000 contains contents of Accumulator
```


STX (STore the X register) Sends the contents of the X register to a specified memory location (see STA). The X register is not affected.

STY (STore the Y register) Sends the contents of the Y register to a specified memory location (see STX). The Y register is not affected.

TAX (Transfer Accumulator to X register) Transfers the contents of the Accumulator to the X register. The Accumulator is not affected.

TAY (Transfer Accumulator to Y register) Transfer the contents of the Accumulator to the Y register. The Accumulator is not affected (see TAX).

TXA (Transfer X to Accumulator) Sends the contents of the X register to the Accumulator. The X register is not affected. Combined with TAY, can be used to transfer a value from X to Y.

Example

```
LDX #$05     ;#$05 in X
TXA          ;#$05 in A
TAY          ;#$05 in Y
```

TYA (Transfer Y to Accumulator) Transfers the contents of the Y register to the Accumulator. The Y register is not affected. Combined with TAX, can be used to transfer a value from Y to X.

Example

```
LDY #$05     ;#$05 in Y
TYA          ;#$05 in A
TAX          ;#$05 in X
```

Index

ABOUT THE AUTHOR

Leonard I. Malkin teaches and does research in biochemistry at a leading midwest medical school. He is a member of the medical school committee on CAI (Computer Assisted Instruction) and has published several commercially successful educational programs for the Apple II. He is also president of his own educational software corporation and is currently engaged in developing other educational programs for the Apple. His immediate plans are to "use the proceeds from this book to buy myself a boat and get away from computers for a while—they're driving me crazy."

6758-5

Hi-Res Graphics and Animation Using Assembly Language

Leonard I. Malkin, Ph.D.

Explore the secrets of double hi-res and color and learn how to create startlingly detailed animated graphics.

This is *the* book for both beginners in assembly language programming and assembly language literates who wish to extend their expertise to Apple II hi-res graphics and animation.

Whereas most books on assembly language do not feature hi-res graphics, this book uses hi-res graphics and animation as a tool to demonstrate the fundamentals of assembly language programming. At the same time, this easy-to-read and sometimes humorous book will show you the secrets of producing professional hi-res animated graphics for games and a variety of other applications.

The Guide for Apple® II Programmers

In the first part of the book, following a discussion of binary and hexadecimal numbering systems and the use of an assembler, you will be guided through each step leading to the construction of a single arcade-type game written entirely in assembly language. Various aspects of hi-res animated graphics are covered with clear, concise explanations. Illustrated with detailed programs, topics include: vertical and horizontal movement, internal animation, paddle and joystick controls, collisions and explosions, scoring, sound generation, and the animation of multiple shapes.

The second part of the book explores color graphics and double hi-res techniques and requirements, including double hi-res color. Other topics in Part Two include curved and diagonal movement, drawing over backgrounds, more sophisticated paddle and joystick controls, integrating BASIC with assembly language, and the use of BASIC and assembly language graphics for educational programs.

Hayden Book Company

A DIVISION OF HAYDEN PUBLISHING COMPANY, INC.
HASBROUCK HEIGHTS, NEW JERSEY / BERKELEY, CALIFORNIA

ISBN 0-8104-6758-5