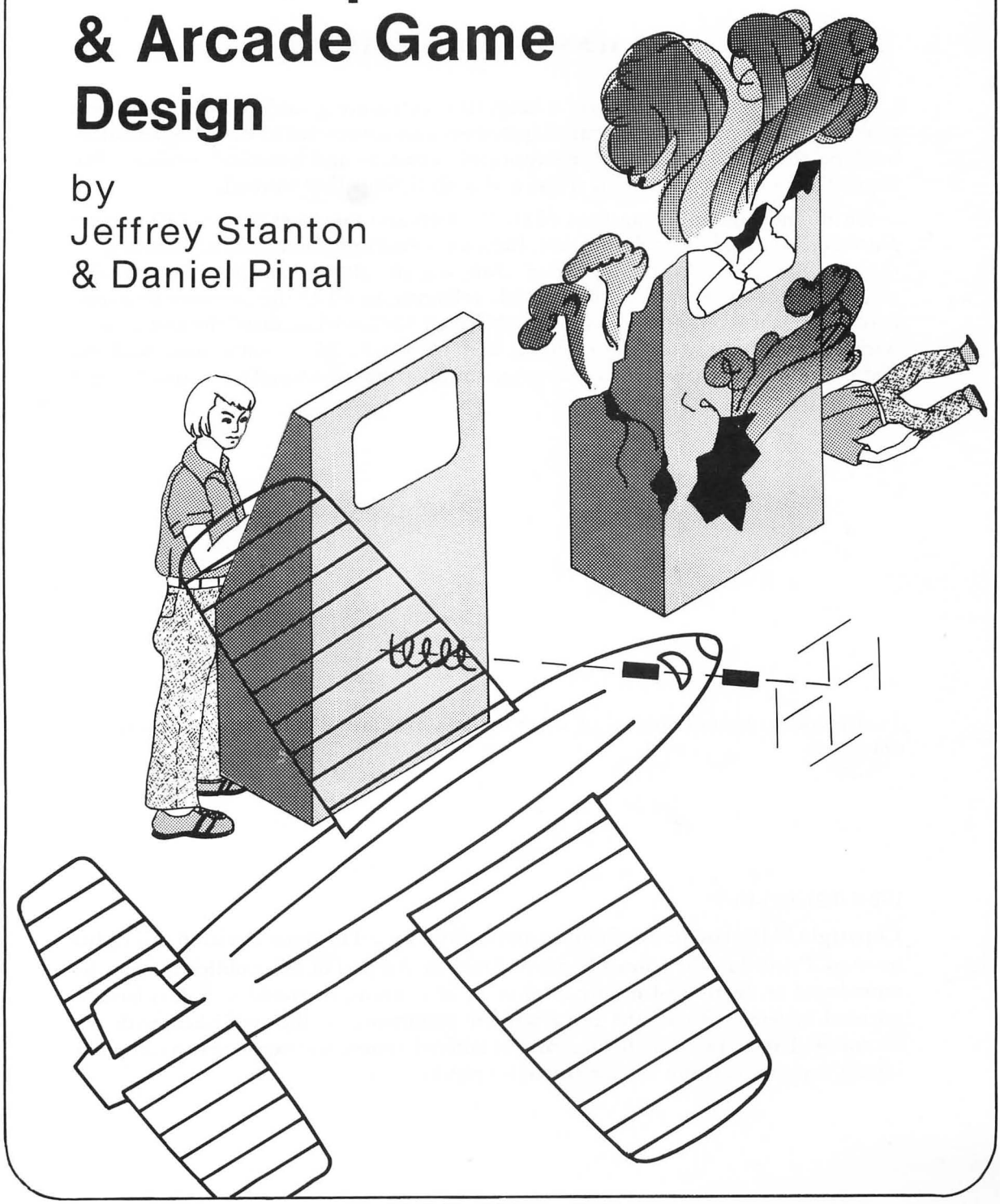
Atari Graphics and Arcade Game Design

By Jeffrey Stanton
with Dan Pinal

Thomas McNair
2315 E. Meadow Creek Dr.
Meridian, ID 83642

Atari Graphics & Arcade Game Design

by
Jeffrey Stanton
& Daniel Pinal



ARRAYS, INC.
The Book Division
11223 S. Hindry Ave. Los Angeles, CA 90045

ACKNOWLEDGEMENTS

A technical book like this was a long, time-consuming undertaking. The book took much longer than we ever anticipated because we wanted to write the definitive book on the subject with both meaningful examples and technical accuracy. We wanted a book that was free of errors and with listings that worked.

We are indebted to the authors of *De Re Atari* and the *Atari Personal Computer Hardware Manual* (Atari Computer, Inc.) who shed light on the inner workings of the computer, and to the author of *Mapping the Atari* (Compute Books) who produced a comprehensive and valuable reference to all of the memory locations inside the Atari computers. Thanks to Michael Mellin who edited the manuscript without meddling with the content, and to Estela Montesinos who without complaint redid the numerous changes in my diagrams again and again until we got it right.

Trademarks, corporate names, or other designations are used for reference purposes only.

ISBN 0-912003-05-7

Copyright © 1984 by Jeffrey Stanton and Arrays, Inc./The Book Division. All rights reserved. Printed in the United States of America. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the prior written permission of the publisher, with the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication.

TABLE OF CONTENTS

Preface

Computer owners, who view stunning graphic effects or play visually exciting games, rarely consider the effort or techniques required to achieve those images. They don't realize that a programmer's ability to create Atari graphics can be compared to an artist's ability using a sketchpad or an animator's skill using animation techniques. In fact, an arcade game is basically an interactive cartoon in which the player controls a character or object that influences the action of the remaining computer-controlled objects on the screen. This action, like in a movie, consists of individual frames viewed at high speed to produce fluid motion. The objects in a game can be animated by either moving them from one screen position to another without changing their shape, or by changing their shapes between frames. In either case, the effect is the same—a feeling of motion.

Atari computers are wonderful graphics machines capable of extraordinary visual effects. Unfortunately, few of these can be implemented directly from Atari BASIC without a thorough knowledge of Machine language and the architecture of the machine. Those who understand the techniques and have mastered them are mostly too busy writing programs to share their knowledge.

This book will allow you to enter the world of Atari graphics in which your most imaginative ideas can be animated. The various chapters will present a comprehensive course in both Atari graphics and high-speed arcade animation techniques. While at least half of the book requires the ability to program in Assembly language, we were careful to begin the book with the simplest graphics concepts in Atari BASIC. The book aims to increase the novice programmer's skill. It assumes no prior knowledge of either Atari graphics or Assembly language. Since we know that many of our readers will be young teenagers, we made every attempt to include BASIC program examples, some with Machine language subroutines, in most of the chapters. We felt that concepts like custom display lists, color indirection, scrolling, character set animation, and player-missile graphics can be learned by beginners, but we didn't neglect the advanced programmer either. We cover the most advanced topics possible on a Machine language level. We discuss vertical blank and display list interrupts, kernals, bit-mapped graphics, sound, scrolling, and player-missile graphics, and use these techniques to develop four complete Assembly language games.

The only requirements for this book are an inquisitive mind, perseverance, and a good Assembler. Although prior Assembly language programming experience isn't necessary, you won't be able to write code without an Assembler.

We will attempt to explain the ideas in this book through a combination of text, drawings, flow charts, and working code. The concepts in this book may seem easy at times, and somewhat difficult at other times. The Atari is a complex machine with many idiosyncrasies. The hardware sometimes makes game design relatively easy, yet the concept of an interrupt-driven machine with its timing problems can make advanced programming frustrating. Our advice is to read the book in stages and try the examples. Learn how they work.

While our goal for presenting the material was to educate a new generation of arcade game designers, I dread the proliferation of copy cat games. The world doesn't need an eighth *Pac-Man* or a tenth *Q*Bert*. They have been done. We hope that programmers both young and old will use their imaginations to create something novel and exciting.

JEFFREY STANTON
VENICE, CALIFORNIA
JULY 14, 1984

PROGRAM LISTINGS AVAILABLE ON DISK

All of the code listed in this book is available on diskette to readers who disdain typing long computer programs. We barely managed to cram all of the listings without DOS on three disk sides, one side BASIC and two sides assembly language. These disks and files are unprotected. We decided to offer readers a choice of buying all three sides, just the BASIC programs, just the Assembly language source code and game object files, or a disk containing all of the games for those who only wish to play them.

The cost of these disks is nominal and can be ordered using the card in the back of the book from Stanton Products, 3710 Pacific Avenue #16, Marina del Rey, California 90292. The prices are as follows:

1) BASIC program listings only	- $10.00
2) Assembly language listings only	- $15.00
3) All program listings (2 disks)	- $20.00
4) Playable games only	- $12.50

Include $1.50 postage. California residents add 6½% sales tax.

The F-S Macro Assembler 40/80, a completely compatible upgrade to *SYNASSEMBLER*, is also available from Stanton Products under a licensing agreement with Funsoft and S-C Software. It is a disk based co-resident assembler, well suited to both beginners and professional programmers. It comes in two versions on the disk; 40 and 80 column achieved entirely through software. Its powerful macro and conditional assembly features makes it one of the most powerful assemblers available for both the newer XL series of Atari computers and the older 400/800 computers. There are 25 pseudo-ops and 31 commands, designed to make life easier for the programmer. With the ability to chain source files and assemble object code to disk, the programmer is only limited by the amount of online disk storage. It is available, as all development tools should be, on an unprotected disk. A 45-page manual is included. It sells for $50.00 plus $1.50 postage and any applicable sales tax. It can also be ordered using the coupon in the back of this book.

CHAPTER 1

GRAPHICS MODES AND COLOR REGISTERS

The ATARI 400/600/800/1200/1400 home computers are some of the most impressive graphics machines available at prices under $1000. Each of these very special machines, in addition to containing a 6502B microprocessor that runs at nearly twice the speed of many competing computers, also contains a number of custom hardware chips. One of these is a separate graphics microprocessor called ANTIC. These powerful chips free the 6502B to do what it was designed to do, calculate.

The computer was built to be extremely flexible with multiple graphics modes, redefinable character sets, indirect color registers, player-missile graphics, collision registers, display list interrupts, fine scrolling, and a built-in sound generator. These features give a polished, smooth, colorful look to the display, almost an arcade look. The effect isn't a coincidence, for the computer is nearly a clone of the hardware used in some of Atari's arcade machines like *Missile Command.*

Skilled Assembly language programmers can harness many of the Atari's capabilities, but less skilled or even beginning programmers may find the machine's flexiblity downright intimidating. Atari BASIC limits the programmer's choices, but it also eliminates much of the error prone graphics initialization that might produce wacky displays. Fortunately, capabilities like player-missile graphics and four-voice sound can still be accessed by PEEKing and POKEing to the machine's hardware addresses.

Most computers limit their graphics display to one or two modes because they are hardware dependent and memory specific. The Apple IIe, for instance, has Lo-Res and Hi-Res graphics. The 8K block of Hi-Res graphics is hard wired to specific memory locations $2000-$3FFF. Each byte (7 pixels) in display memory contains both pixel and color information that hardware uses to raster color dots on the television screen. If you move an object, you must change all of the bytes at both the old and new locations in screen memory without erasing the background. Worse yet, if you want to change an entire blue screen to red, you need to rewrite all 8K bytes of screen data. Similarly, scrolling requires a memory shuffle and at best results in slow, jerky motion.

The Atari, on the other hand, uses the ANTIC graphics microprocessor to interpret the graphics data in screen memory and another chip, the CTIA/GTIA, to plot the color information. Each of the fourteen different graphics modes (six text/character and eight graphic) uses differing amounts of screen memory for display. As a rule of thumb, the higher the resolution and the more colors available for display, the more memory required. Essentially, you need one bit of information for each pixel displayed plus additional bits for color information. Required screen memory, including the accompanying display list, can be as little as 261 bytes for double

width, double height text, or as much as 7891 bytes for a hi-resolution 320 x 192 pixels screen. The screen can be placed nearly anywhere in memory as long as you tell ANTIC where to obtain its data.

The programmer can mix graphics modes because he can instruct ANTIC to display data from a specific part of memory in a particular graphics mode. This means you can have medium resolution graphics at the top of the screen, text in the middle and hi-resolution graphics at the bottom, if you like. The display is stacked in horizontal bands that stretch across the entire width of the screen. Any combination of display modes can be choosen as long as the entire display does not exceed 192 scan lines.

You are not even limited to using just the graphics characters in the ROM character set for your playfield graphics. A new character set can be customized with a character set editor. The computer will substitute the new set when you set the character set pointer to its address.

The unique way that the Atari computer handles color information through a series of addressable color registers gives the programmer added flexiblity. Each screen pixel is assigned to one of four playfield color registers when it is plotted. Simply changing the color value in one of these registers changes the color of every pixel assigned to that register. An entire background can be changed from red to blue with one simple POKE.

Although you can choose from 128 colors, you can work with only four at a time. Fortunately, you can gain extra colors either by adding different colored players to the display, or by changing the colors in mid-display using display list interrupts. Moreover, the addition of three GTIA modes allows either sixteen colors with one luminance, or one color with sixteen luminances, or nine colors. Although these modes sacrifice resolution to some extent, they are useful for three-dimensional shading, or adding a lot of color to the screen easily.

Perhaps the most powerful and useful graphics feature on the Atari is the player-missile graphics. Since these objects are completely independent of playfield memory, they can move smoothly over the display without affecting it. If the programmer wishes, he can set the priority so that one or more of the four players (five without missiles) can pass behind other players or playfield objects. Each player can be of different color, height and width, with a maximum width of eight pixels. In addition each player and each missile has individual collision registers that keep track of any collisions between each other and the playfields.

The Atari computer was the first home computer with player-missile or sprite graphics. It unfortunately doesn't allow true X,Y positioning. While the horizontal position can be set with a single POKE, the player's data must be shifted within a 128 or 256 (depending on the resolution) block of player memory to obtain true vertical positioning. In addition, although players can be plotted in double or quadruple width, the basic width of eight pixels imposes further limitations on the programmer.

Certainly, the use of player-missile graphics makes programming smooth animated graphics light years easier than on non-sprite-oriented computers like the

Apple. But some of the newer computers like the Commodore 64 have eight players, each 24 x 21 pixels with true X,Y positioning, and both Coleco's Adam and the Texas Instrument's TI-99A have thity-two sprites, each 32 x 32 pixels with true X,Y positioning. While it may be easier to program these computers using players exclusively, each has only three or four graphics modes that can't be mixed, and none can smooth-scroll the playfield without extensive memory shuffling.

The Atari's ability to fine scroll the graphics display either horizontally or vertically sets the Atari apart from all other home computers. The Atari does this easily because the ANTIC chip can begin with data anywhere in memory and automatically map to the screen sequential memory bytes on a line by line basis. To vertically rough scroll a 40-character per line text screen you only need to change by 40 bytes the start of the memory area from which ANTIC gets display data. You can scroll the screen less, just a few scan lines at a time or a portion of one character, by setting the vertical fine scroll register. Horizontal scrolling is done similarly, but the data structure is much more difficult to set up because the data lines do not follow sequentially in memory but have gaps to store the off-screen image. To move any row requires only resetting the memory pointer by one byte; however, the reset must be performed for each row or mode line that you want to scroll.

Since the Atari computer is interrupt driven, time critical program code can be executed during the computer's vertical blank period at a rate of exactly 60 times a second. It is also the perfect time to scroll the screen, move players, and check collisions, for discontinuous screen jumps can't occur when the electron beam is between frames.

The Atari computer has many powerful graphics features that overall far surpass any home computer on the market today. We will explain each of these features in greater detail through working arcade game examples in the appropriate chapters in this book.

Display Modes

The Atari computer's fourteen display modes includes six text or character graphics modes, and eight bit-mapped graphics modes of varying resolution. In addition, computers containing the GTIA color chip have three extra graphics modes which are special cases of the highest resolution bit-mapped graphics mode.

Programmers using the Atari BASIC cartridge from non-XL machines can easily access only nine of these fourteen display modes. Although XL machines can access four of these five additional modes directly from BASIC, these extra modes, which include several multi-colored character set modes requiring custom character sets, are best left to the Assembly language programmer.

The character graphics modes 0, 1, and 2 are easy to understand. The normal text display is graphics 0. It consists of twenty-four rows of forty characters. Each character is eight dots wide by eight dots or television scan lines high. Graphics mode 1 characters are just graphics mode 0 characters stretched twice as wide; only twenty of these characters fit on any row. Graphics 2 characters are as wide as

graphics one characters but are twice as deep. Since each character is sixteen scan lines high, only twelve rows would fit on the screen in the full screen mode.

Normally a text window occupies the bottom four rows (32 scan lines) in all BASIC graphics modes except the GTIA modes. If you were to set the display for graphics mode 2 characters, ten rows of these characters would be displayed above four rows of graphics 0 characters. The display can be set to full screen by adding 16 to the graphics mode. Thus, invoking GRAPHICS 2+16 sets up a full screen display consisting of twelve rows of graphics 2 characters.

Graphics modes 3 thru 8 are bit-mapped graphics modes that don't involve characters. Instead they plot colored dots or pixels of varying resolution. A pixel can be as large as eight dots by eight dots in graphics mode 3, be four dots by four dots in graphics mode 5, two dots by two dots in graphics 7, or be as small as one single dot in graphics mode 8. The finest resolution in graphics mode 8 gives us complete control over every dot on the screen.

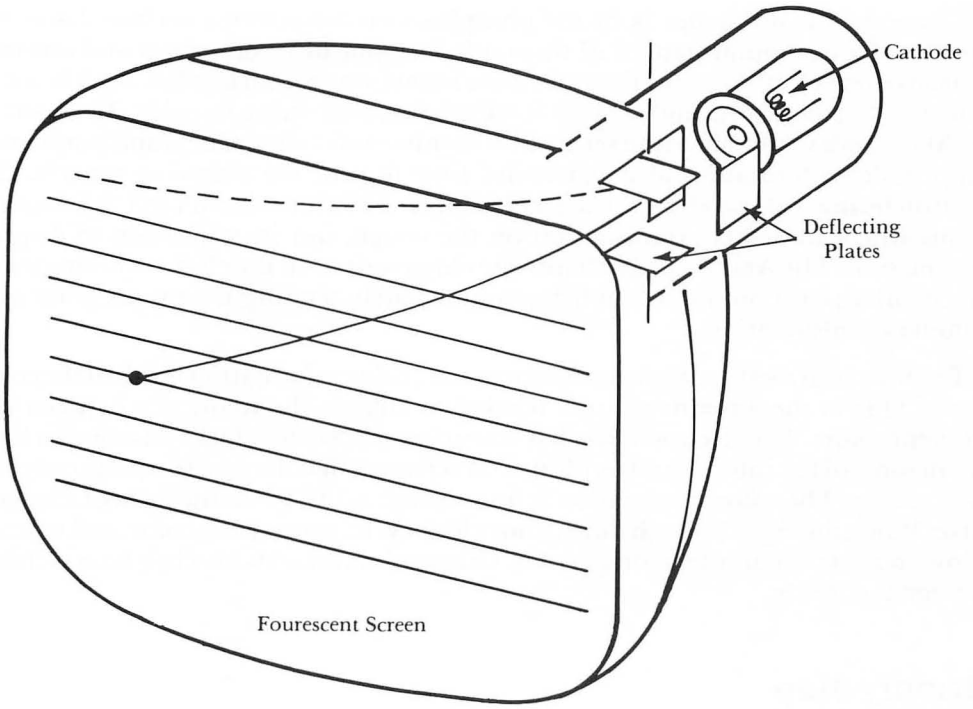
Obviously, there are more reasons to use each of these modes beyond the varying degrees of resolution. As the graphics modes increase in resolution, more memory is required for display. Bigger pixels simply fill up the screen faster and therefore require less memory. For example, a graphics 3 screen that has pixels the size of a character and a screen resolution of forty pixels by twenty-four, requires only 480 bytes for screen memory. A graphics 8 screen that stores groups of eight pixel-sized dots in one byte, requires 7680 bytes of screen memory. A programmer using this mode on a 16K Atari 400 or 600XL would have very little room for his program.

The number of colors available is another reason for choosing a particular graphics mode. As a rule, more memory is required to display more color because the color information must be encoded within the screen data. For example, the only difference between the two color graphics modes 4 and 6, and the four color modes 3, 5, and 7 is in the amount of memory required. Graphics modes 4 and 5 have the same resolution (80 x 48), but graphics mode 4 uses only half as much memory. Graphics mode 5 must use two bits to encode the color for each pixel, where only one bit is needed to tell the computer how to display graphics mode 4 pixels. A similar relationship exists between graphics modes 6 and 7.

You have probably noticed that each graphics mode is displayed in a series of rows. A full-screen graphics 5 screen consists of forty-eight vertically stacked rows of pixels, each four scan lines high. A graphics seven screen consists of ninety-six vertically stacked rows of pixels, each two scan lines high. In each case there are 192 scan lines displayed on the screen. This number is no accident but is determined by the way television sets draw or scan a picture.

How Televisions Work

Most television sets, including the ones on which you actually watch TV, are raster scan devices. A moving electron beam strikes the individual phosphors that are painted on the inside front surface of the television tube. Each time an electron strikes a phosphor, it glows momentarily. If the beam continues to strike the same phosphor it will glow continously.



In order to draw an entire screen full of glowing dots the electron beam has to move in a series of accurate sweeps across the picture tube. A charged deflection plate bends the electron beam so that the stream of electrons strikes a series of adjacent phosphors along a straight horizontal line. These closely packed individual dots appear to be a solid line. The electron beam starts at the top of the screen and scans from left to right. When it finishes a line, it shuts down briefly while it returns to the left edge and drops down to the next scan line. This period is known as the "horizontal blank." The electron beam does this 192 times in a process known as "raster scan." When the beam finishes it returns to the top of the screen in a time period known as "vertical blank."

Obviously, if the pixels are to remain lit for longer than a brief moment the screen will have to be refreshed quite often. The electron beam retraces its 192 line path sixty times a second. A television set actually scans 262 scan lines, but the average set can only display slightly more than 200 lines. The area above and below your rectangular playfield contains some of these extra lines.

In order to get an image other than solid white, the electron beam's intensity is varied while it is scanning. By varying the intensity or the number of electrons that hit individual phosphors, different brightness levels are achieved. This gives us shading on black and white television sets that ranges from black through various shades of grey to pure white.

The process of producing a color image is only slightly different. The electron beam scans as usual from left to right over the playfield's 192 scan lines at sixty times

per second. The difference is in the phosphors on the screen's surface. Each dot consists of a triangular pattern of three sub dots; one blue, one green and one red. Instead of one electron beam there are three beams, one for each color. Each beam is aimed very precisely through a mask at the subdots containing its color. Thus, when the Atari sends to the television set or color monitor color (frequency) and luminance (amplitude) information at a particular time during the scanning process, the electron beams will produce the desired display. The Atari doesn't tell the electron beams where to display information on the screen, but instead when to display information. The Atari's special hardware chips wait until the electron beams reach a particular point on the screen before immediately sending the proper color and luminance information.

The time unit used to determine when to send color information is called the color clock. This is the amount of time it takes to change the frequency between the different colors. The electron beam has time to send 228 color clocks on one scan line. Again some of the information is plotted off screen so that the Atari displays only 160 color clocks. The color information is transmitted to the TV in the form of a square wave. When the signal is high during one clock cycle, you get one color, and when it is low you get the complementary color. Other colors are obtained by phase shifting the signal slightly.

Memory Map

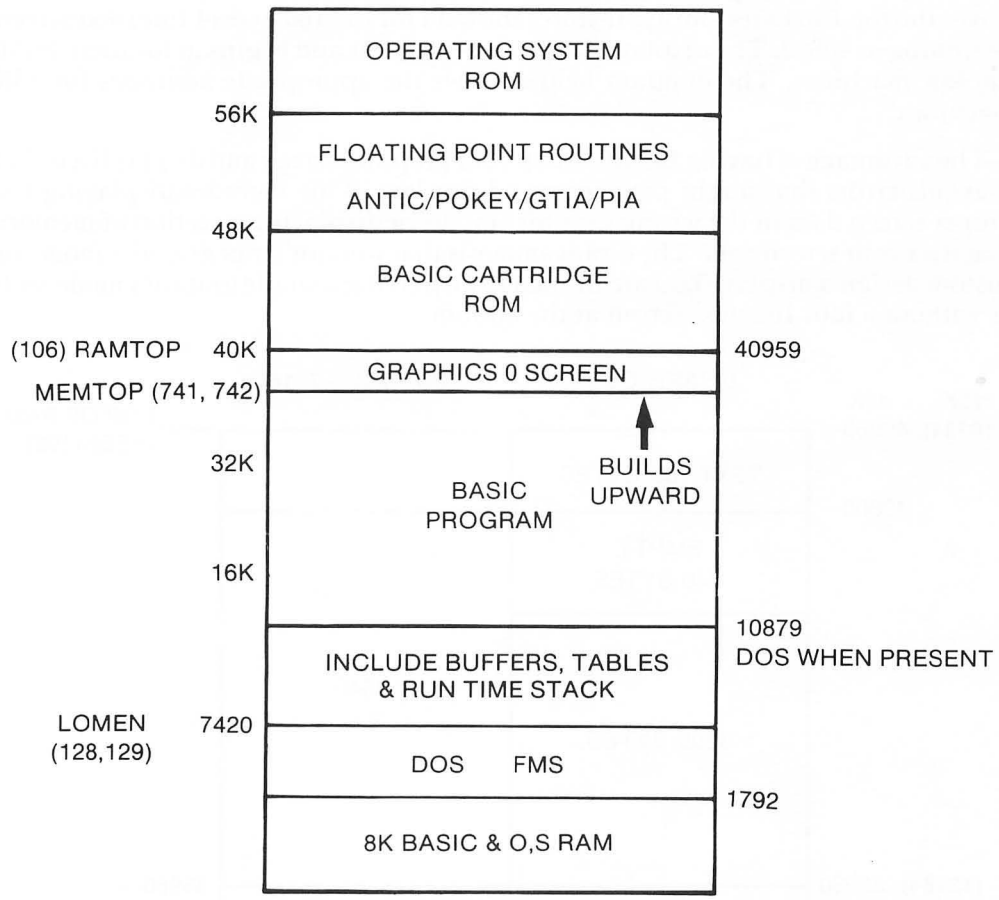
It is best at this time to develop an understanding of where the Atari computer stores your BASIC computer program and the location of display memory. All 6502-based Atari computers, address 64K bytes of memory whether physical memory exists or not. The computer is divided into RAM (Random Access Memory) and ROM (Read Only Memory). You can store things like your BASIC program in RAM memory. Atari 400's and 600XL's contain 16K of RAM memory, while the larger 800's, 1200's and 1400's contain 48K or 64K of RAM memory. Most of the area above 48K, from 52K to 64K is reserved for the computer's Operating System (OS), and the use of various hardware chips ANTIC, POKEY, CTIA/GTIA and PIA. The OS and all of these chips are in ROM. Locations in them can be read, but only a very few hardware locations in some of the chips can be written to. Since many of these writable hardware locations can't hold information for longer than 1/60 of a second, they are "shadowed" by RAM memory locations in the lower portion of the computer's memory. These "shadowed" memory locations, that contain information such as joystick/paddle values for each of the registers, are copied to the appropriate RAM locations during the vertical blank period between television frames. Color information "shadowed" in RAM is copied to the hardware registers at the same time. This not only saves you the trouble of refreshing the hardware each cycle, but allows you to read the "shadowed" values.

The lowest section of memory contains zero page, BASIC and OS system RAM, the stack, and the keyboard buffer. All of this occupies the space between 0 and 1535 ($000-$5FF). The area between 1536 and 1791 ($600-$6FF) known as page six is free for user Machine language subroutines. The area above 1792 to location 7420 or LOMEM is reserved for the DOS File Management System. LOMEM drops to 1792

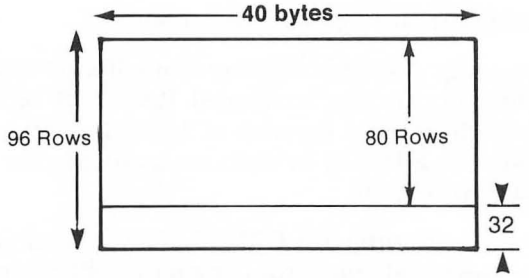
in computers that don't use a disk drive or interface module.

The 8K ROM BASIC cartridge occupies memory from 40K to 48K regardless of memory configuration. When the cartridge is engaged, RAMTOP or HIMEM drops to 40959 in 48K and 64K machines and remains at 16384 in 16K machines. The memory between LOMEM and HIMEM is the area available for running and storing your BASIC language programs.

BASIC programs are stored beginning at LOMEM. The program along with its buffers, tables, and run time stack build upwards in memory. When a graphics mode is invoked, BASIC reserves the area just below HIMEM for the graphics screen and text window if there is one. It stores a small program called the display list just below the graphics screen. This display list, which we will discuss in greater detail in the next chapter, tells ANTIC where to find display data and in which graphics mode to display it. In brief,the list contains an instruction for each row of graphics data, plus



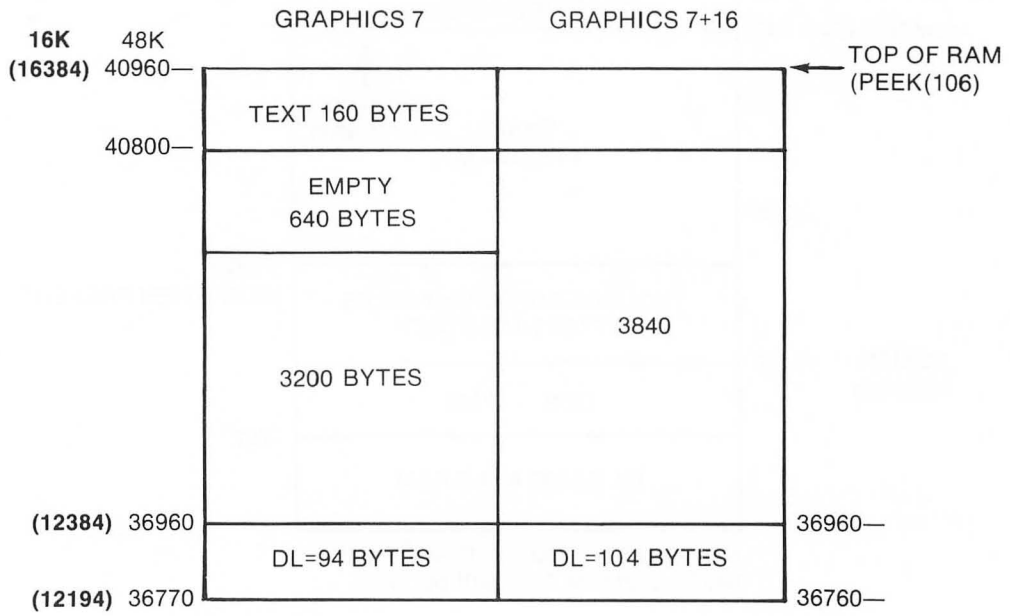
NOTE: 1536—1792 FREE Ram Pg. 6
NOTE: LOMEM 3341 without DOS



instructions about the number of blank lines to skip before plotting plus the locations of the screen's data and the beginning of its own display list.

For example, if we set up a full screen graphics 7 screen (40 x 96), BASIC places the beginning of screen memory automatically at 36960 for a 48K machine. The screen's memory is 3840 bytes. The 104-byte display list is placed at memory location 36760. If we choose instead a graphics 7 screen with a text window (40 x 80), only 3200 bytes of screen memory are required. BASIC puts screen memory at the same location but leaves the top 640 bytes empty. It stores the data for the 160 byte (4 line) text screen beginning at 40800. The display list is 10 bytes shorter and begins at location 36770 for 48K machines. The diagram below shows the appropriate addresses for 16K machines.

The advantage of having BASIC set up your graphics screen and display list is that it avoids errors that might produce weird displays. This includes displaying the proper screen data in the wrong graphics mode, or displaying a section of memory that isn't your screen data. The disadvantage is that you can't mix graphics modes or custom design a display. You are limited to displaying a single graphics mode with or without a four line text screen at the bottom.



Color

The Atari uses a very flexible method to display color. Instead of storing the color of each pixel directly in display memory, the Atari refers the color information to a specific color register. Each pixel has the color register number stored rather than a specific color. This method, is extremely flexible, but it allows only a maximum of five colors on the screen at any one time. In an effort to save screen memory at most only two bits are used to specify a pixel's color. On the other hand, if you wish to change the background color or the color and luminence of all the pixels referring to a particular color register, you need only to change the color value in that one color register.

There are five available color registers numbered 0 through 4. Color register 4 is also known as the background color register because it specifies the color and luminance for any place on the screen where nothing else is written. In the bit-mapped graphics modes 3-8, this means the color between any of the plotted pixels. In text mode 0 it means the border area, not the color behind the character.

The color registers are each one byte long. The upper four bits determine the color (0-15), and the lower four specify the brightness. Only the highest three of the four luminance bits are used, so that there are eight levels of brightness. Sixteen different colors and eight levels of brightness create 128 shades of color. The arangement for each color is in groups of sixteen. Values 0-15 are for color #0 in different intensities, 16-31 for color #1, etc. The table below lists the values for many of the common colors.

As we said, there are five different color registers. Think of these as paint pots. You can put a color into any one of these color registers and then draw points (pixels) and lines using that color register. It is similar to drawing on a canvas with a brush. The difference is that if you draw a green line five pixels long with color register 0, screen memory doesn't store it as green, green, green, green, green, but as a series of bits 01 01 01 01 01. When ANTIC fetches screen data and feeds it to the CTIA/GTIA chip, this color chip looks to the appropriate color register to determine which color is to be put on the screen for each separate pixel. For most of the four color graphics modes the bit pattern is as follows:

00 BACKGROUND
01 REGISTER #0
10 REGISTER #1
11 REGISTER #2

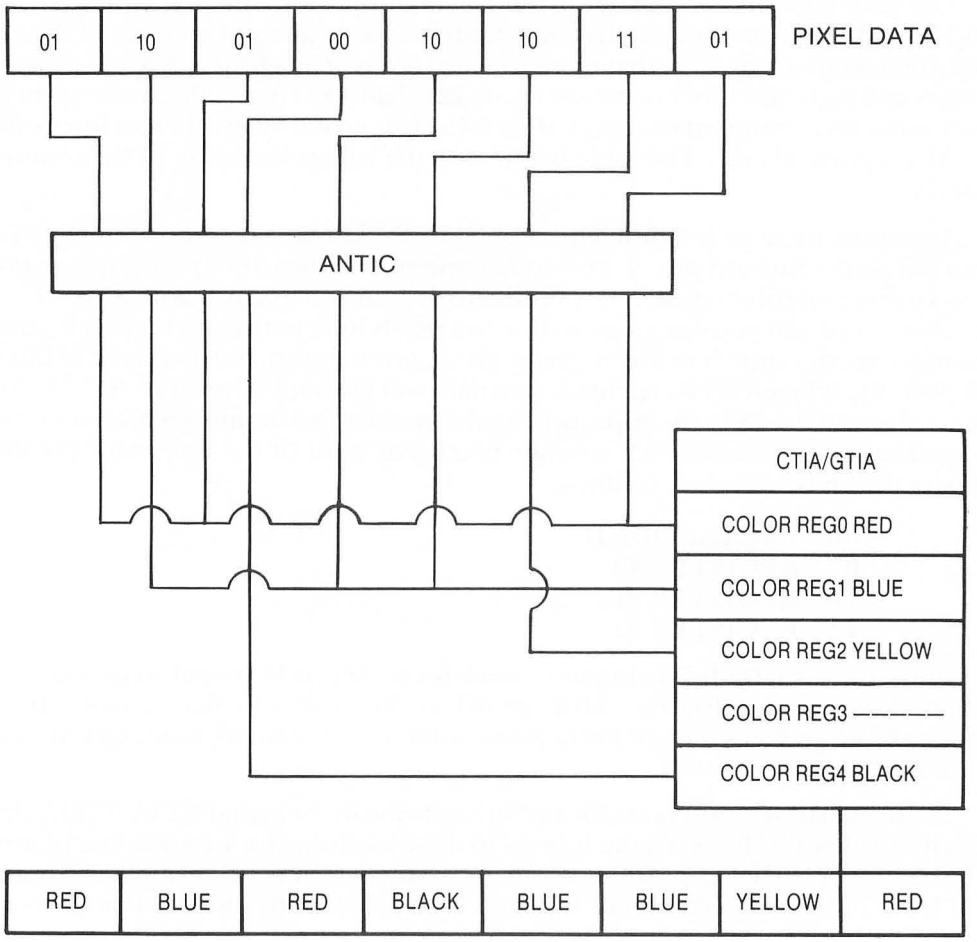
Thus, if we had the following screen data for eight pixels, the colors drawn by the color chip would reflect the values stored in the different color registers. In the example below, black is in the background color register and red, blue and yellow are in the other color registers.

These five color registers are located in hardware in the actual CTIA/GTIA chip. Each time ANTIC feeds it data, it looks to these hardware locations before plotting the color. Even what appears to be a blank empty screen is still generated from the CTIA/GTIA's interpretation of the data. Even all zero bits indicate that the entire

screen is just background. The chip looks to these registers thousands of times during the refresh process of updating the screen.

The operating system also maintains copies of these color registers in RAM memory. These are called shadow color registers. They are maintained because the hardware locations are "write only" locations. Since they can't be read, we need RAM locations where they can be read. At the beginning of each refresh cycle, these five shadowed registers are copied into the hardware locations.

	CTIA REGISTER		O.S. SHADOW REG.	
PLAYFIELD #0 /COLOR REG #0	708	($2C4)	53270	($D016)
PLAYFIELD #1 /COLOR REG #1	709	($2C5)	53271	($D017)
PLAYFIELD #2 /COLOR REG #2	710	($2C6)	53272	($D018)
PLAYFIELD #3 /COLOR REG #3	711	($2C7)	53273	($D019)
PLAYFIELD #4 /COLOR REG #4	712	($2C8)	53274	($D01A)
(BACKGROUND)				



BASIC uses the SETCOLOR command to set up the color registers. It is in the form of SETCOLOR (color reg #), (color #),(luminance #). A direct POKE to the O.S. shadow register is equivalent to SETCOLOR and is faster. For example SETCOLOR 1,3,8 is the same as POKE 709,(3*16)+8 or POKE 709,56.

COLOR VALUES FOR COLOR REGISTERS

	VALUE	HUE	LUMI-NANCE		VALUE	HUE	LUMI-NANCE
BLACK	0 ($00)	0	0	MAGENTA	82 ($52)	5	2
DARK GREY	4 ($04)	0	4	PURPLE	96 ($60)	6	0
GREY	6 ($06)	0	6	LAVENDER	102 ($66)	6	6
WHITE	14 ($0E)	0	14	BLUE	116 ($74)	7	4
GOLD	20 ($14)	1	4	LT BLUE	120 ($78)	7	8
YELLOW	24 ($18)	1	8	TURQUOISE	164 ($A4)	10	4
BROWN	34 ($22)	2	2	GREEN	196 ($C4)	12	4
TAN	36 ($24)	2	4	LIGHT GREEN	200 ($C8)	12	8
ORANGE	54 ($36)	3	6	YELLOW GREEN	214 ($D6)	13	6
RED	66 ($42)	4	2	OLIVE GREEN	228 ($E4)	14	4
PINK	72 ($48)	4	8	PEACH	246 ($F6)	15	6

Atari BASIC's COLOR command, used to specify a particular playfield register that plots points or draws lines, is probably the most confusing aspect of Atari graphics. When you choose a COLOR #, it selects a color register assigned to a playfield. It uses that color register to plot with until a new color register is chosen. The problem is that the COLOR # often doesn't correspond to the color register #, and varies with the graphics mode.

There is a logical explanation for the discrepancy, but it is more apparent to the Assembly language programmer than to the casual BASIC programmer. Remember that in most modes two bits are used to specify the color. This is true in all of the bit-mapped graphics modes. Color #0 is usually background because a Machine language 00 written into display memory usually plots nothing. If COLOR # is 1 it writes a 01 in display memory for that pixel, if equal to a 2 it writes a 10, and if equal to a 3 it writes an 11. Unfortunately, if you refer to the table above, bits 01 corresponds to color register #0, bits 10 to color register #1, and bits 11 to color register #2. On the other hand, the two color modes use only COLOR #1 to plot points because just a single bit is used to direct the CTIA to the color register.

The character graphics modes, 0, 1, and 2 are even more confusing. The upper two bits in each character not only determine which color register is selected, but effect which character is displayed. Essentially, parts of the character set appear in different colors. For example, if you are using the computer's default colors and you are in character mode #1, a 20 character per line mode, upper case letters appear in orange, lower case in light green, inverse upper case characters in dark blue, and inverse lower case graphics characters in red.

	ATASCII	COLOR REGISTER
Uppercase alphabet (A-Z) numbers, punctuation	39-90	0
Lowercase alphabet	61-122	1
Inverse uppercase alphabet numbers, punctuation	160-218	2
Inverse lowercase alphabet	225-250	3

Since the relationships between the COLOR # and the playfield it refers to differ in many of the graphics modes, beginners will do best by refering to the table below or the one in their BASIC reference manual. The concept of drawing lines and shapes using color registers as paint pots is best illustrated with the following demonstration. We will draw in graphics mode 7 full screen. This is a four color mode with three foreground colors and one background color. Initially, we will fill our paint pots or color registers with dark gray, green, blue and red.

COLOR REG. #0	GREEN	COLOR #1
COLOR REG. #1	BLUE	COLOR #2
COLOR REG. #2	RED	COLOR #3
COLOR REG. #4	DARK.GREY	COLOR #0

We will draw our rectangular paint pots at the bottom of the screen and one shape in its color above each pot. We will also draw another shape or line above one of the other paint pots. When we are finished we will have six shapes above three paint pots. The solid shapes are filled in using the standard XIO color fill command in BASIC.

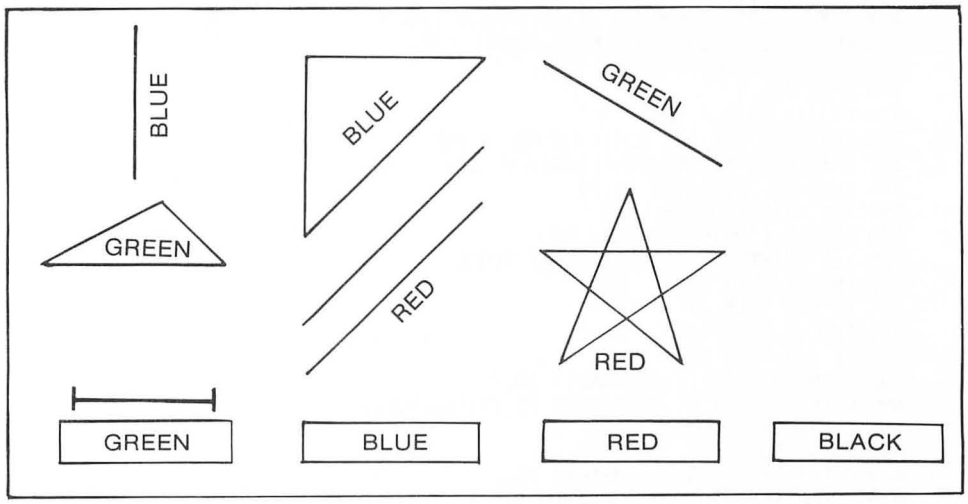
The XIO fill command is designed to work with four-sided figures, but if you are careful it will work with triangles. In general you plot a point in the lower right hand corner of the figure and use DRAWTO statements to reach the upper right hand corner of the figure. You next use the position statement to move the cursor to the lower right hand corner and use the POKE statement to place a number, equal to the COLOR number to be used for plotting into memory location 765. Last, you perform a XIO 18, #6,0,0,"S". The fill commands works from left to right and will fill the figure until it encounters any illuminated pixel between the left and right sides of the figure being filled. For example:

```
20 GRAPHICS 7+16
50 SETCOLOR 1,7,2:REM BLUE
80 COLOR 2
140 PLOT 35,90:DRAWTO 35,80:DRAWTO 5,80
150 POSITION 5,90
160 POKE 765,2
170 XIO 18,#6,0,0,"S"
180 GOTO 180
```

When you fill a triangle, two sides of the figure are drawn. While you may not be at the top of the figure when you finish, you will be on the left side when you reposition the cursor to the bottom right. Usually, the color fill works properly; but, if you look at the blue triangle, you will notice that we repositioned the cursor at the bottom one pixel to the left. This is because when the last line fills on the bottom it must have a right boundary to fill to. If the boundaries are equal, the line will begin filling from there to the right edge of the screen.

The colored bar above the paint pot indicates which paint pot or color register can be adjusted by the joystick. You can select an individual paint pot with the select key. When the bar is to the far right beyond the last paint pot, it points to the paint pot used to color the background. You can adjust the color in the paint pot by moving the joystick up or down. If you begin at the first paint pot, which is intially green, and change the color, you will notice that the two shapes that were green changed to the new color in the paint pot. It doesn't matter if the shape is above the paint pot or somewhere else on the screen. What matters is which color register or paint pot was in effect when the shape was drawn, for those pixels contain data that point to a particular color register.

You can play with the four paint pots and watch the various shapes change color. When you change the background color one or more of the other shapes will vanish if the two paint pots are identical. The shape is still there, but the pixels instruct the CTIA/GTIA chip to produce the same color as the background. The shape just blends in to become invisible.



```
10 REM COLOR PAINT POT DEMO
20 GRAPHICS 7+16
30 SETCOLOR 4,0,2:REM DARK GREY
40 SETCOLOR 0,12,2:REM GREEN
50 SETCOLOR 1,7,2:REM BLUE
60 SETCOLOR 2,4,2:REM RED
70 REM DRAW SCREEN IN THREE COLORS
80 COLOR 1
90 PLOT 95,10:DRAWTO 135,30
100 PLOT 35,50:DRAWTO 30,40
110 POSITION 5,50
120 POKE 765,1
130 XIO 18,#6,0,0,"S"
140 PLOT 35,90:DRAWTO 35,80:DRAWTO 5,80
150 POSITION 5,90
160 POKE 765,1
170 XIO 18,#6,0,0,"S"
180 REM DRAW IN SECOND COLOR
190 COLOR 2
200 PLOT 55,31:DRAWTO 75,15:DRAWTO 55,15
210 POSITION 55,30
220 POKE 765,2
230 XIO 18,#6,0,0,"S"
240 PLOT 20,10:DRAWTO 20,35
250 PLOT 80,90:DRAWTO 80,80:DRAWTO 50,80
260 POSITION 50,90
270 POKE 765,2
280 XIO 18,#6,0,0,"S"
290 REM DRAW IN THIRD COLOR
300 COLOR 3
305 PLOT 55,65:DRAWTO 75,45
308 PLOT 55,60:DRAWTO 75,40
310 PLOT 130,70:DRAWTO 115,45:DRAWTO 100,70
315 DRAWTO 135,55:DRAWTO 95,55:DRAWTO 130,70
320 PLOT 130,90:DRAWTO 130,80:DRAWTO 100,80
330 POSITION 100,90
340 POKE 765,3
350 XIO 18,#6,0,0,"S"
351 REM DRAW INITIAL POSITION COLOR BAR
352 C=1:COLOR 1:CSET=INT(PEEK(708)/16)
354 PLOT 10,75:DRAWTO 30,75
360 REM MAIN PROGRAM LOOP
370 IF PEEK(53279)<>5 THEN 500
380 REM SHIFT COLOR BAR TO NEXT BLOCK
382 IF C=1 THEN 390
384 IF C=2 THEN 410
386 IF C=3 THEN 430
388 IF C=4 THEN 450
390 COLOR 0:PLOT 10,75:DRAWTO 30,75
400 COLOR 1:PLOT 55,75:DRAWTO 75,75:C=2:POT=1
402 CSET=INT(PEEK(709)/16)
405 GOTO 500
410 COLOR 0:PLOT 55,75:DRAWTO 75,75
420 COLOR 1:PLOT 105,75:DRAWTO 125,75:C=3:POT=2
422 CSET=INT(PEEK(710)/16)
425 GOTO 500
430 COLOR 0:PLOT 105,75:DRAWTO 125,75
440 COLOR 1:PLOT 140,75:DRAWTO 159,75:C=4:POT=4
442 CSET=INT(PEEK(712)/16)
445 GOTO 500
450 COLOR 0:PLOT 140,75:DRAWTO 159,75
460 COLOR 1:PLOT 10,75:DRAWTO 30,75:C=1:POT=0
```

```
462 CSET=INT(PEEK(708)/16)
500 FOR DE=1 TO 5:NEXT DE
510 REM CHANGE COLOR PAINT POT WITH JOYSTICK
520 IF STICK(0)<13 OR STICK(0)=15 THEN 700
530 IF STICK(0)=13 THEN 600
540 REM SHIFT SELECTED PAINT POT COLOR
550 CSET=CSET+1:IF CSET=16 THEN CSET=0
560 SETCOLOR POT,CSET,2
570 GOTO 700
600 CSET=CSET-1:IF CSET<0 THEN CSET=15
610 SETCOLOR POT,CSET,2
700 FOR DE=1 TO 20:NEXT DE:GOTO 370
```

Graphics Modes

ATARI computers can display fourteen graphics modes of which nine can be directly accessed from BASIC on older machines and thirteen on the newer XL units. This section of the book will explain each of the various graphics modes, their resolution or size, the method by which they are mapped to the screen, and how colors are generated.

Graphics Mode 0 (ANTIC 2)

This is the normal-sized character or text mode that the computer defaults to on start up. Being a character mode, screen memory consists of bytes that represent individual characters in either the ROM or a custom character set. ANTIC displays forty of these 8 x 8 sized characters on each of twenty-four lines.

Graphics 0 is a 1½ color mode. Color register #2 is used as the background color register. Color register #1 sets the luminance of the characters against the background. Setting the color has no effect. Bits within a character are turned on in pairs to produce the luminace color. Otherwise single bits tend to produce colored artifacts on the high resolution screen. These colors depend on whether the computer has a CTIA or GTIA chip, and the color of the background.

Graphics 1 (ANTIC 6)

This is one the expanded text modes. Each characters is 8 x 8 but the pixels are one color clock in width instead of the 1/2 color clock mode of Graphics 0 making the characters twice as wide. Only twenty characters fit on any line. A graphics 1 screen has twenty rows while the full screen mode has twenty-four rows of characters.

The two high bits of each ATASCII character, that normally identify lowercase or inverse video text in Graphics 1, set the color register for the 64 character set. Decimal character numbers 0-63 use color register zero, while those same 64 characters if given character numbers 64-127 use color register #1. If you are typing from the Atari keyboard, the uppercase letters A-Z ATASCII 65-90 (Internal # 33-58) are assigned to color register zero, while the lowercase numbers 97-122 (Internal # 97-122) are assigned to register #1.

GRAPHICS MODES

	GRAPHICS MODE	ANTIC MODE	DISPLAY TYPE	AVAILABLE COLORS	SCREEN SIZE Columns x Rows	SCAN LINES MODE
TEXT MODES	0	2	Standard Text	1 Color & 2 Luminances	40 x 24	8
	1	6	Double-Width Text	5	20 x 20 (Split) 20 x 24 (Full)	8
	2	7	Double-Width Double-Height Text	5 5	20 x 10 (Split) 20 x 12 (Full)	16
PIXEL MODES	3	8	FOUR COLOR GRAPHICS	4	40 x 20 (Split) 40 x 24 (Full)	8
	5	A		4	80 x 40 (Split) 80 x 48 (Full)	4
	7	D		4	160 x 80 (Split) 160 x 96 (Full)	2
	4	9	TWO COLOR GRAPHICS	2	80 x 40 (Split) 80 x 48 (Full)	4
	6	B		2	160 x 80 (Split) 160 x 96 (Full)	2
	8	F	High Resolution Graphics	1 Color 2 Luminances	320 x 160 (Split) 320 x 192 (Full)	1
GTIA MODES	9	—	16 Luminance Medium Resolution	1 Color 16 Luminance	80 x 192 (Full)	1
	10	—	9 Color Medium Resolution	9	80 x 192 (Full)	1
	11	—	16 Color Medium Resolution	16	80 x 192 (Full)	1
CHARACTER	12*	4	Multi-Color Character	4	40 x 20 (Split) 40 x 24 (Full)	8
	13*	5	Double High Multi-Color Char.	4	40 x 10 (Split) 40 x 12 (Full)	16
BIT MAPPED	14*	C	Two Color Bit-Mapped	2	160 x 160 (Split) 160 x 192 (Full)	2
	15*	E	Four Color Bit Mapped	2	160 x 160 (Split) 160 x 192 (Full)	2

* BASIC modes on XL machines only

	BYTES/ LINE	MEMORY USED (Bytes)	COLOR REGISTER NUMBERS FOREGROUND	BACKGROUND	BORDER	COLOR SHADOW REGISTER NUMBER	REGISTER
	40	992	1(color is not selectable)	2	4	—	—
	20	674 672	0,1,2,3	4	4	See Table	—
	20	424 420	0,1,2,3	4	4	See Table	—
	10	434 432	0,1,2	4	4	Color 0 Register 4	712
	20	1174 1176	0,1,2	4	4	Color 1 Register 0	708
	40	4190 4200	0,1,2	4	4	Color 2 Register 1 Color 3 Register 2	710 709
	10	694 696	0	4	4	Color 0 Register 4	712
	20	2174 2184	0	4	4	Color 1 Register 0	708
	40	8112 8138	1 (color is not selectable)	2	4	Color 0 Register 2 Color 1 Register 2	710 709
	40	8138	4	—	—	Color 0-15= Luminance Register 4=Color	712
	40	8138	1-8	0	0	Set Registers By Pokes	712
	40	8138	0-15	—	—	Color 0-15=Color Register 4= Luminance	712
	40	1154 1152	0,1,2,3	4	4	Register 0 Register 1 Register 2	708 709 710
	40	664 660	0,1,2,3	4	4		
	20	4270 4296	0	4	4	Register 0	708
	40	8112 8138	0,1,2	4	4	Register 0 Register 1 Register 2	708 709 710

Graphics 2 (ANTIC 7)

This text mode is basically the same as the previous mode except that each row of pixels is two scan lines high. Thus 12 rows of 20 characters are displayed on a full screen. Only ten rows fit on a split screen.

Graphics 3 (ANTIC 8)

This four-color graphics mode turns a split screen into 20 rows of 40 graphics cells or pixels. Each pixel is 8 x 8 or the size of a normal character. The data in each pixel is encoded as two bit pairs, four per byte. The four possible bit pair combinations 00, 01, 10, and 11 point to one of the four color registers. The bits 00 is assigned to the background color register and the rest refer to the three foreground color registers. When the CTIA/GTIA chip interprets the data for the four adjacent pixels stored within the byte, it refers to the color register encoded in the bit pattern to plot the color.

Graphics 4 (ANTIC 9)

This is a two-color graphics mode with four times the resolution of GRAPHICS 3. The pixels are 4 x 4, and 48 rows of 80 pixels fit on a full screen. A single bit is used to store each pixel's color register. A zero refers to the background color register and a one to the foreground color register. The mode is used primarily to conserve screen memory. Only one bit is used for the color, so eight adjacent pixels are encoded within one byte, and only half as much screen memory is needed for a display of similiar-sized pixels.

Graphics 5 (ANTIC A or 10)

This is the four color equivalent of GRAPHICS 4 sized pixels. The pixels are 4 x 4, but two bits are required to address the four color registers. With only four adjacent pixels encoded within a byte, the screen uses twice as much memory, about 1K.

Graphics 6 (ANTIC B or 11)

This two color graphics mode has reasonably fine resolution. The 2 x 2 sized pixels allow 96 rows of 160 pixels to fit on a full screen. Although only a single bit is used to encode the color, screen memory still requires approximately 2K.

Graphics 7 (ANTIC D or 13)

This is the four color equivalent to GRAPHICS mode 6. It is the finest resolution four color mode and naturally the most popular. The color is encoded in two bit

pairs exactly the same way as in GRAPHICS 3. The memory requirements of course is much greater as there are 96 rows of 160 — 2 x 2 sized pixels. It requires 3840 bytes of screen memory with another 104 bytes for the display list.

Graphics 8 (ANTIC F or 15)

This mode is definitely the finest resolution available on the Atari. Individual dot-sized pixels can be addressed in this one-color, two-luminance mode. There are 192 rows of 320 dots in the full screen mode. Graphics 8 is memory intensive; it takes 8K bytes (eight pixels/byte) to address an entire screen.

The color scheme is quite similar to that in GRAPHICS mode 0. Color register #2 sets the background color. Color register #1 sets the luminance. Changing the color in this register has no effect, but, this doesn't mean that you are limited to just one color.

Fortunately, the pixels are each one half of a color clock. It takes two pixels to span one color clock made up of alternating columns of complementary colors. If the background is set to black, these columns consist of blue and green stripes. If only the odd-columned pixels are plotted, you get blue pixels. If only the odd-columned pixels are plotted, you get green pixels. And if pairs of adjacent pixels are plotted, you get white. So by cleverly stagering the pixel patterns, you can achieve three colors. This method is called artifacting. This all depends on background color and luminance.

The following five graphics modes have no equivalent in BASIC on older machine but if indicated do correspond to an equivalent graphics mode on the newer XL models.

Antic 3

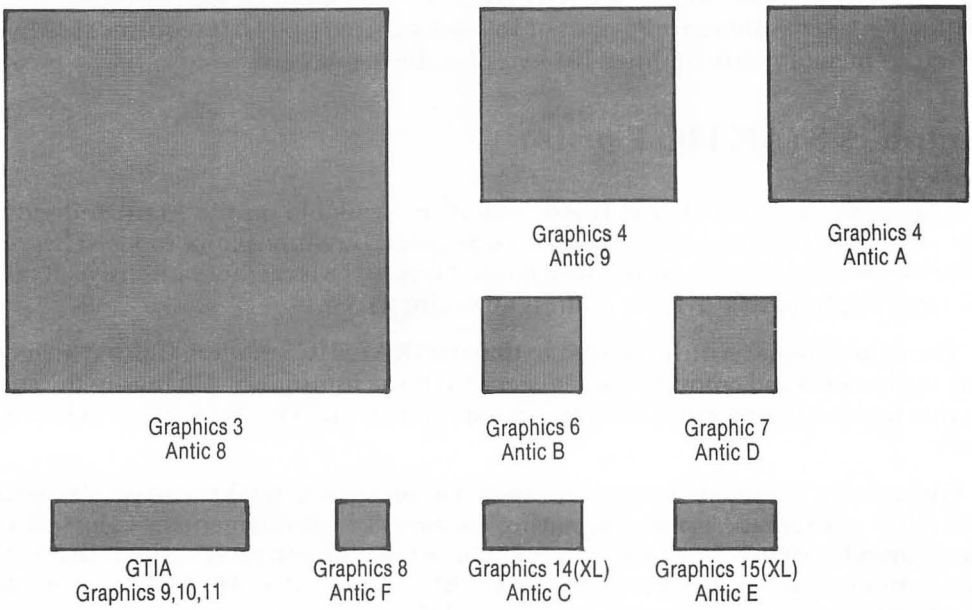
This rarely used text mode is sometimes called the lowercase descenders mode. Each of the forty characters per line are ten scan lines high, but since each of the characters are only eight scan lines high, the lower two scan lines are normally left empty. However, if you use the last quarter of the character set, the top two lines remain blank, allowing you to create lowercase characters with descenders.

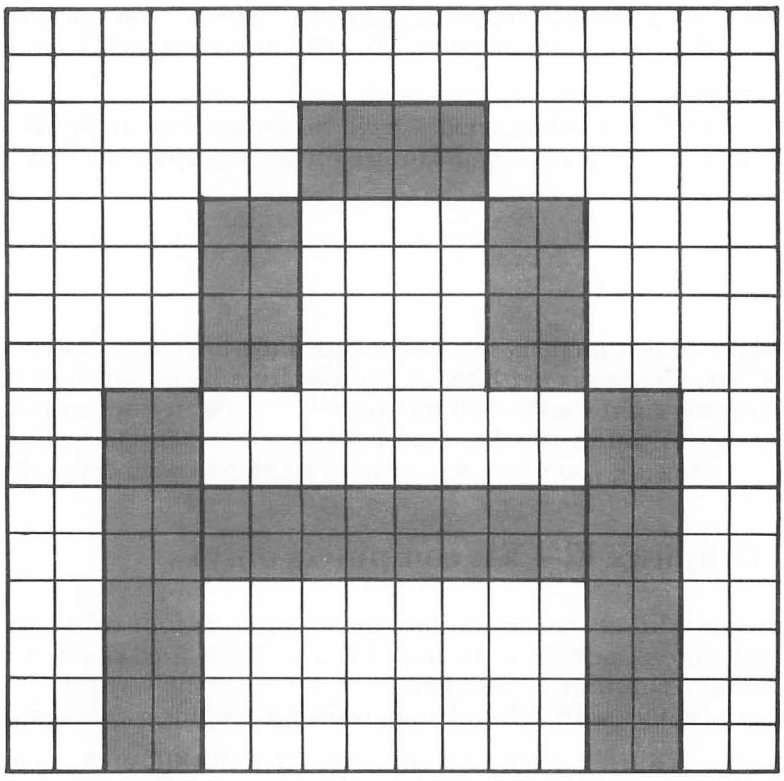
Antic 4 (Graphics 12—XL computers only)

This very powerful character graphics mode supports four colors while using relatively little screen memory (1K). In addition its 4 x 8 sized characters have the same horizontal resolution as GRAPHICS 7, yet twice the vertical resolution. A large number of games with colorful and detailed playfields use this mode.

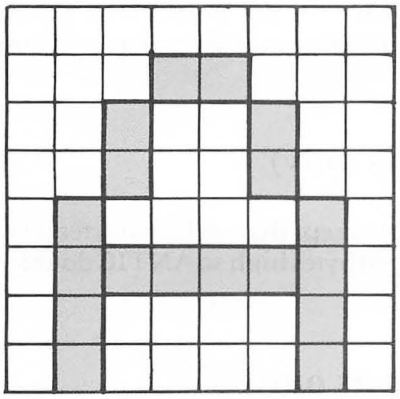
These characters differ considerably from ANTIC 6 (BASIC 2) characters, in that each character contains pixels of four different colors, not just a choice of one color determined by the character number. Each byte in the character is broken into four

RELATIVE PIXEL SIZES OF DIFFERENT GRAPHIC MODES

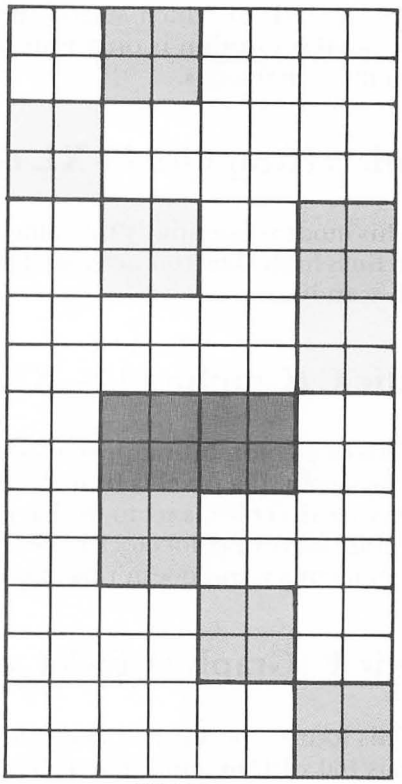




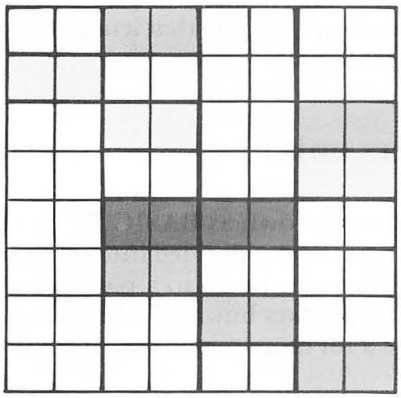
GRAPHICS 2
ANTIC 7



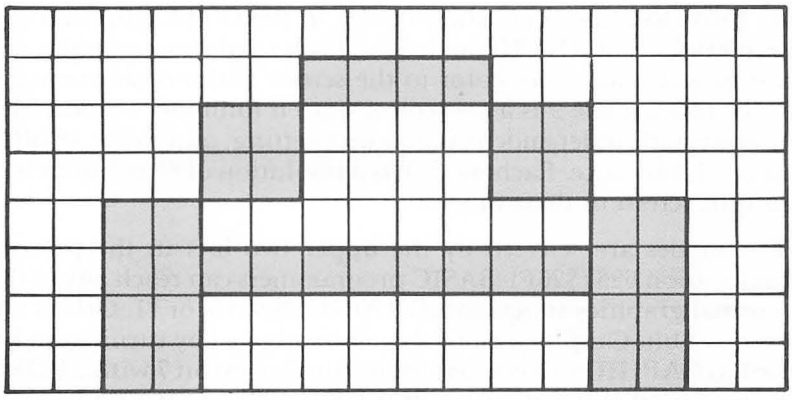
GRAPHICS 0
ANTIC 2



GRAPHICS
13 (XL)
ANTIC
5



GRAPHICS 12 (XL)
ANTIC 4



GRAPHICS 1
ANTIC 6

bit pairs, each of which selects the color register for the pixel. That is why the horizontal resolution is only four bits. A special character set generator is used to form these characters.

Antic 5 (Graphics 13-XL computers only)

This mode is essentially the same as ANTIC 4 except that each character is sixteen scan lines high. The character set data is still eight bytes high so ANTIC double plots each scan line.

Antic C (Graphics 14—XL computers only)

This two-color, bit-mapped mode the eight bits correspond directly to the pixels on the screen. If a pixel is lit it receives its color information from color register #0, otherwise the color is set to the background color register #4. Each pixel is one scan line high and one color clock wide. This mode's advantages are that it only uses 4K of screen memory and doesn't have artifacting problems.

Antic E (Graphics 15-XL computers only)

This four-color, bit-mapped mode is sometimes known as BASIC 7½. Its resolution is 160 x 192 or twice that of GRAPHIC 7. Each byte is divided into four pairs of bits. Like the character data in ANTIC 4, the bit pairs point to a particular color register. The screen data, however, is not character data but individual bytes. The user has a lot more control, but this mode uses a lot more memory, approximately 8K.

GTIA

There are three additional graphics modes in the GTIA chip that are actually special interpretations of ANTIC mode $F, a high resolution graphics mode. They are designed to add a lot more color to the screen without sacrificing too much resolution. Graphics mode 9 is a one-color, sixteen luminance mode. Mode 10 is a nine-color mode with independent luminance setting, and mode 11 offers sixteen colors set at one luminance. Each mode has a resolution of 80 columns by 192 rows. There is no split screen in these modes.

The GTIA modes are selected by the upper two bits in the priority register shadowed at location 623 ($26F). BASIC programmers can reach any of these GTIA modes by normal graphics statements GRAPHICS 9, 10, or 11. Others will need to POKE the correct bit. Graphics mode 9 can be activated by turning on bit 6 with a POKE 623,64. GRAPHICS 10 is activated by turning on bit 7 with a POKE 623,128. Both bits 6 and 7 need to be set with a POKE 623,192 to activate graphics mode 11. These values will disturb the other bits in GPRIOR that set various functions, so

take care if you have set any other bits previously by POKEing a value that combines both. These other bits allow the combination of all four missiles into a fifth player, establish player-missile and playfield priorities, and enable multiple-colored or overlapping players.

The GTIA chip was not standard equipment on Atari computers until December, 1981. Those with older computers may wonder if this chip is installed. If you are on the text screen (GR.0) and do a POKE 623,64, the screen will go black and become unreadable with the GTIA chip. If nothing happens, you have the CTIA chip. Your machine can be updated to the newer GTIA chip by your Atari dealer if you desire.

Since GTIA modes allow a lot more color while using the same screen memory as GRAPHICS 8, more bits are needed to keep track of the color. In fact sixteen colors require four bits, so that only two pixels are encoded within a byte instead of the usual eight. This is the reason the horizontal resolution drops from 320 pixels to 80 pixels per scan line. The pixels are elongated instead of square. Also, since there are only nine color registers in the computer, the sixteen colors are bit mapped to the screen as in other computers, instead of by the Atari method of color indirection.

Graphics 9

GRAPHICS mode 9 produces up to sixteen different luminances of the same hue. This is quite useful for drawing pictures that require alot of shading, or for digitizing pictures. The main color is set by the background color register #4. You can use the SETCOLOR command to set the color value in the upper four bits (nybble), and the luminance in the lower four bits to zero. The COLOR command is used to vary the luminance. What actually happens is that the pixel data from ANTIC is logically ORed with the lower nybble of the background color register to set the luminance that appears on the screen. A quick little program that will demonstrate the mode is listed below.

```
10 GRAPHICS 9
20 SETCOLOR 4,1,0
30 FOR I=0 TO 15
40 COLOR I
50 PLOT I+20,10
60 DRAWTO I+20,40
70 NEXT I
80 GOTO 80
```

Graphics 11

GRAPHICS 11 is a one luminance, 16 color mode. The luminance this time is set by the background color register #4. The SETCOLOR command is used to set up the single luminance value in the lower nybble of this register, while zeros representing the hue are placed in the upper nybble. The COLOR command is used in this mode to select the various colors. This time ANTIC's pixel data is logically ORed with the

upper nybble of the background color register to set the hue that appears on the screen. A typical example follows:

```
10 GRAPHICS 11
20 SETCOLOR 4,0,6:REM LUMINANCE = 6
30 FOR I=0 TO 15
40 COLOR I
50 PLOT I+20,10
60 DRAWTO I+20,40
70 NEXT I
80 GOTO 80
```

This sixteen-color mode doesn't use the four playfield color registers for color indirection. The colors on the screen are determined directly by the data bits stored in memory. Each pixel has the value for the hue stored in memory. For example, a blue hue, which is number 7, has a bit value 0111. This is what is stored in the nybble. Two adjacent pixels, which are stored in the same byte have a value of 01110111 or 119 decimal ($77). All have the luminance assigned to the background color register.

There is a slight advantage to not using the color registers in either GRAPHICS 9 or 11: even more color can be added to the screen by adding players of different colors. The disadvantage is that the collision registers are useless in both modes.

Graphics 10

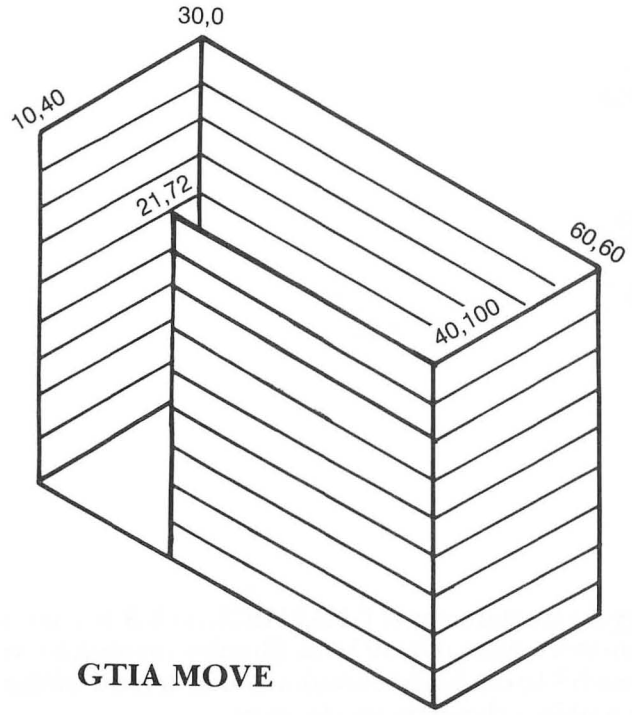
GRAPHICS 10 is probably the most versatile of the GTIA modes. While it only has nine colors with variable luminance, it does use color indirection to produce its color. This means that the screen data points to one of the five playfield-color registers and the four player-color registers to obtain its color information rather than storing the color data on the screen as in the other two GTIA modes. Since players use the same color as part of the background, you must be careful. They will blend in when they coincide with that portion of the playfield using the same color. Collision registers don't work in this mode either. We believe ANTIC's collision registers are bypassed in all GTIA modes. A typical GRAPHICS 10 example follows;

```
10 GRAPHICS 10
20 FOR I=0 TO 8
30 N=I
40 POKE 704+I,N*16+6
50 COLOR I
60 PLOT I*2+20,10
70 DRAWTO I*2+20,30
80 NEXT I
90 POKE 704,0:REM FOR BLACK BACKGROUND
100 GOTO 100
```

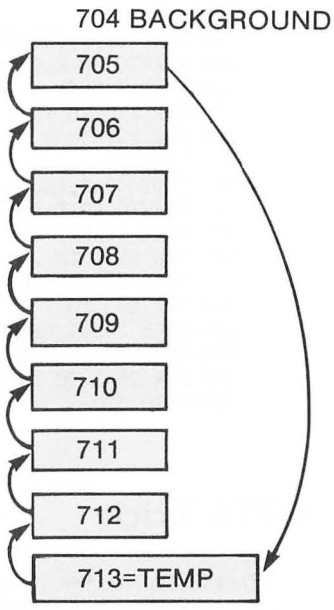
When you choose the color register via the COLOR statement in BASIC, 16 different values can be used but only the first nine are valid. The remainder just repeat various playfield registers. The chart is listed below. You will notice that COLOR 4 no longer sets the background color. Instead it is set by player 0's color in register number 704. It is best to use POKEs to set the color registers rather than SETCOLOR.

COLOR STATEMENT	REGISTER #	PLAYFIELD
0	704	PCOLOR0
1	705	PCOLOR1
2	706	PCOLOR2
3	707	PCOLOR3
4	708	PLAYFIELD 0
5	709	PLAYFIELD 1
6	710	PLAYFIELD 2
7	711	PLAYFIELD 3
8	712	PLAYFIELD 4

You can achieve some really nice animation effects when using this mode through the power of color indirection. If you don't count the background, you can rotate eight colors through the color registers to create a sense of motion.



GTIA MOVE



ROTATE COLOR REGISTERS

The example below draws a very colorful rectangular box in perspective. Part of one side was left open so that you can see more of the left and back sides. The different color registers or paint pots are set up in a bucket brigade. Since location 713 isn't used for anything, we used this as a temporary storage location. The color or paint from register 705 is first put into this temporary bucket or storage location, then shift the rest of the colors by moving them from the higher color register to the next lower one. This is done in a FOR...NEXT loop. We POKE the lower color register with the value we find in the next higher color register.

```
10 GRAPHICS 10
20 REM LOAD COLOR REGISTERS
30 POKE 704,0:POKE 705,82
40 POKE 706,116:POKE 707,196
50 POKE 708,54:POKE 709,68
60 POKE 710,24:POKE 711,102
70 POKE 712,34
80 FOR K=0 TO 19
90 FOR I=0 TO 8
100 COLOR I
110 PLOT 10+K,(40-K*2)+I*10
120 DRAWTO 10+K,(50-K*2)+I*10
130 NEXT I:NEXT K
150 FOR K=0 TO 29
160 FOR I=0 TO 8
170 COLOR I
180 PLOT 30+K,(K*2)+I*10
190 DRAWTO 30+K,(10+K*2)+I*10
200 NEXT I:NEXT K
210 FOR K=0 TO 19
220 FOR I=1 TO 8
230 COLOR I
240 PLOT 60-K,(60+K*2)+I*10
250 DRAWTO 60-K,(70+K*2)+I*10
260 NEXT I:NEXT K
270 FOR K=0 TO 19
280 FOR I=1 TO 8
290 COLOR I
300 PLOT 40-K,(100-K*2)+I*10
310 DRAWTO 40-K,(110-K*2)+I*10
320 NEXT I:NEXT K
325 FOR DE=1 TO 200:NEXT DE
330 POKE 713,PEEK(705)
340 FOR I=0 TO 7
350 POKE 705+I,PEEK(706+I)
360 NEXT I
370 FOR DE=1 TO 15:NEXT DE
380 GOTO 330
```

GTIA Trick

The GTIA modes, because they are special cases of GRAPHICS mode 8, require a considerable amount of display memory, approximately 8K. Graphics mode 0, a text mode, in many respects is very much like our high resolution mode. It is a 1 ½ color mode, and the individual pixels within a character are the same.

If you turn on GTIA mode 11 from GRAPHICS 0 by POKING 623,192 the screen turns black and you get weird colored pixel patterns where your characters were. Recall from the above discussion that the color pixel patterns are directly bit mapped in screen memory in pairs of four bits, or nybbles, two per byte. If we could rewrite the character set so that the sixteen possible nybble pairs were in the first sixteen characters in the set, we could plot colored blocks the size of characters. While it wouldn't be as fine a resolution as in normal GRAPHICS 11, it would only require 960 bytes of screen memory, quite a substantial savings.

The bit pattern that is used to set color registers is the same one that we need to set up our pair of nybbles in each row of our character. The chart is listed below.

BITS	VALUE	COLOR
0000	0	grey (no color)
0001	1	light orange
0010	2	orange
0011	3	red orange
0100	4	pink
0101	5	purple
0110	6	purple blue
0111	7	blue
1000	8	blue
1001	9	light blue
1010	10	turquoise
1011	11	blue green
1100	12	green
1101	13	yellow green
1110	14	orange green
1111	15	light orange

We need to POKE a $00 into each of the bytes of the 0th character, a $11 (decimal 34) into the bytes for the 1st character, a $22 (decimal 34) into the bytes of the third character, etc. The values are seventeen apart so that it is simple to put the correct color nybble values into an array CT(16). Then it is a straightforward affair to method of POKEing the values in eight at a time into the proper character position in the new character set.

Any of these special GTIA color characters can be plotted to the screen at a specific position by calculating the offset from the beginning of screen memory. This location is stored at locations 88 and 89.

```
SCREEN = PEEK(88)+PEEK(89)*256
OFFSET = (40 x row #) + column #
LOCATION = SCREEN + OFFSET
```

So if you want to plot a purple pixel at character location (2,2) you do a POKE SCREEN + 82, 85. The luminance is set by the background color register 712.

An example of putting a row of sixteen different-colored GTIA pixels in GRAPHICS 0 is shown below.

```
10 REM DEMO OF GTIA COLORS IN GRAPHICS MODE 0
15 REM MOVE TOP MEMORY DOWNWARD TO FIT CHARACTER SET
20 POKE 106,PEEK(106)-4
30 CB=PEEK(106):REM NEW CHARACTER SET LOCATION-HI BYTE
40 GRAPHICS 0
50 CHRSET=CB*256:REM ACTUAL RAM LOCATION OF CHARACTER SET
55 REM SET UP ARRAY OF SPECIAL CHARACTER VALUES
60 DIM CT(16)
70 FOR I=0 TO 15:CT(I)=17*I:NEXT I
80 REM WRITE GTIA COLOR CHARACTERS (0-15) IN NEW CHARACTER SET
90 FOR I=0 TO 15
100 FOR J=0 TO 7
110 POKE CHRSET+I*8+J,CT(I)
120 NEXT J:NEXT I
140 REM FIND SCREEN MEMORY & WRITE CHARACTERS
141 SCREEN=PEEK(88)+PEEK(89)*256
150 FOR I=0 TO 15:POKE SCREEN+I,I:NEXT I
160 POKE 756,CB:REM NEW CHARACTER SET LOCATION - HI BYTE
170 POKE 623,192:REM TURN ON GTIA MODE 11 (16 COLOR)
180 POKE 712,8:REM CONTROLS LUMINANCE
190 GOTO 190
```

CHAPTER 2

DISPLAY LISTS

We introduced you briefly in chapter 1 to a graphics microprocessor called ANTIC that is capable of displaying any of fourteen graphics modes. Since any screen actually consists of a collection or vertical stack of these individual graphics modes, ANTIC looks to a program called the display list to determine in which graphics mode it should display the screen data. The fact that there is something resembling a graphics display instruction set makes the computer extremely flexible. It becomes possible to display any collection of graphics modes from data in screen memory that can be stored virtually anywhere within the computer's RAM memory. This flexibility allows the user to mix graphics modes and even scroll the screen in any direction by altering the portion of screen memory displayed.

ANTIC, like most true microprocessors, has an instruction set that is used to write the display list program. The display list specifies three things: where the screen data is located, what display modes to use to interpret the screen data, and what special display options, if any, to implement.

Antic Instruction Set

ANTIC has a simple instruction set with only four basic instruction types. There are map mode instructions, character mode instructions, blank line instructions, and jump instructions. Map mode instructions instruct ANTIC to display a mode line as colored pixels, while character mode instructions tell ANTIC to display a mode line with character data either from its internal ROM or from your own custom designed set. Blank line instructions instruct ANTIC to display a number of horizontal scan lines with solid background color. Like GOTO statements in BASIC, jump instructions change the value in ANTIC's program counter so that it looks for its next opcode somewhere else.

Special Options or Modifiers

ANTIC also has a number of special options or modifiers to its map and character mode instructions. These are specified by setting one of four high bits in the instruction set. These options are load memory scan (LMS), display list interrupt (DLI), vertical scroll, and horizontal scroll.

The load memory scan option is the most frequently used option for it occurs at least once in every display list. It specifies where the screen data is stored in memory.

Technically, only one of these instructions is actually needed in any series of display modes because screen memory is usually continuous. However, if memory isn't continuous, either within a particular display mode or at the boundry between different modes, an LMS instruction is needed each time a new section of memory is used. Another instance where an additional LMS instruction is required, is in ANTIC modes E and F (GRAPHICS 8) where continuous screen memory crosses a 4K boundary. The load memory scan option is invoked by adding a decimal 64 ($40) to the map or character in the mode instruction. This is equivalent to setting the sixth bit in the instruction. LMS instructions are three bytes long. The first byte is the opcode specifying the mode and the last two bytes contain the address of screen memory in low byte, high byte order.

The other three modifiers are sometimes used by Assembly language programmers to achieve special effects. Setting bit 7 or adding 128 to the opcode enables a display list interrupt. Execution of this instruction causes ANTIC to force the 6502 to generate an interrupt. The interrupt service routine will be at the address pointed to in memory locations 512,513 decimal ($200, 201). Display list interrupts are often used to change the colors in the color registers over part of a screen or to change between character sets midway down the screen. We will discuss these uses in detail in chapter 6 and 4, respectively.

Horizontal scrolling can be set up by adding 16 to the opcode or setting bit 4. Likewise, you enable vertical scrolling by adding 32 to the opcode or by setting bit five. These modifiers allow you to fine scroll the screen in either direction. Naturally if you are planning to scroll through memory by changing the start of screen memory, you will need to combine your scroll modifier with a load memory scan modifier. This technique will be shown in more detail in chapter 7.

GRAPHICS MODE INSTRUCTIONS

Mode Dec ↓	Mode Hex ↓	LMS Only Dec	LMS Only Hex	Vertical Scrolling Only Dec	Vertical Scrolling Only Hex	LMS with Horiz. Scrolling Dec	LMS with Horiz. Scrolling Hex	LMS with Horiz. & Vert. Scrolling Dec	LMS with Horiz. & Vert. Scrolling Hex	Comment
2	2	66	42	34	22	82	52	114	72	Text Mode 0
3	3	67	43	35	23	83	53	115	73	10-Scan Line Character
4	4	68	44	36	24	84	54	116	74	Multicolored Character
5	5	69	45	37	25	85	55	117	75	Double High Multicolored Character
6	6	70	46	38	26	86	56	118	76	Text Mode 1
7	7	71	47	39	27	87	57	119	77	Text Mode 2
8	8	72	48	40	28	88	58	120	78	Graphics Mode 3
9	9	73	49	41	29	89	59	121	79	Graphics Mode 4
10	A	74	4A	42	2A	90	5A	122	7A	Graphics Mode 5
11	B	75	4B	43	2B	91	5B	123	7B	Graphics Mode 6
12	C	76	4C	44	2C	92	5C	124	7C	Two-Color Bit Mapped
13	D	77	4D	45	2D	93	5D	125	7D	Graphics Mode 7
14	E	78	4E	46	2E	94	5E	126	7E	Four-Color Bit Mapped
15	F	79	4F	47	2F	95	5F	127	7F	Graphics Mode 8

Note:

1) Display List Interrupts can be enabled by adding 128 decimal, $80 Hex to above values

2) Since it is impractical to do horizontal scrolling without a LMS instruction values are not referenced.

Blanking Instructions

The blanking instructions generate a certain number of blank scan lines in the color and luminance of the background or border color. From 1 to 8 blank scan lines can be generated by these opcodes. They are primarily used to correct the overscan on a television set.

Jump Instructions

There are two jump instructions. The first (JMP) tells ANTIC to continue looking for instructions at a different address. It is equivalent to a GOTO in the display list. It is a three byte instruction with the address in low byte, high byte order following the opcode. Its only function is to provide a solution to the display list's inability to cross a 1K boundary. If for any reason your display list must cross a 1K boundary, then it must use a JMP instruction. Otherwise, don't worry about this instruction.

The second jump instruction (JVB)-Jump and wait for Vertical Blank-is used in every display list. It is a three byte instruction; the address in low byte, high byte order follows the opcode. JVB tells ANTIC to jump to the start of the display list and wait for a new screen refresh to begin. Surprisingly, this address doesn't have to be accurate, because the OS keeps track of the top of the display list and passes it to ANTIC during the vertical blank. However, you should try to maintain the real address because if you use any SIO functions such as the disk drive or the printer, ANTIC won't be updated properly and the jump will be to the address that you specify in the instruction.

ANTIC INSTRUCTION SET

Instruction		Comment
Decimal	Hex	
0	0	1 Blank Line
16	10	2 Blank Lines
32	20	3 Blank Lines
48	30	4 Blank Lines
64	40	5 Blank Lines
80	50	6 Blank Lines
96	60	7 Blank Lines
112	70	8 Blank Lines
1	1	Jump to Location
65	41	Jump & Wait for VBlank

Typical Graphics 0 Display List

Let us look at the display list for a typical Graphics #0 screen, the standard text mode. If you look at the chart below, you will see that this is ANTIC mode 2. It is a character mode with forty bytes per line. Each row is eight scan lines high, and there are twenty-four rows of characters. To produce an entire screen of text, twenty-four mode lines of ANTIC mode 2 will be required.

In BASIC the display list is setup automatically in memory just below screen memory. The memory pointers to the top of the display list are shadowed at locations 560, 561 decimal in low byte, high byte order. Thus:

DLIST = PEEK(560)+256*PEEK(561)

It is possible to look at the display list if, we write a short program to print the entire list to either the screen or a line printer. Display lists are easy to view if there are at least four lines of text 0; but viewing display lists for full screen graphics modes, and especially custom graphics modes, can be a problem without a printer. The reason is that you must PEEK the values in the display list while you are in the appropriate mode, and not when you return to text mode 0. A method to avoid the problem is to store the PEEKed display list elsewhere in memory and print it after you return to the text mode. A safe spot depending on circumstance might be 256 or more bytes below the desired display list.

The following program will print the display list for GRAPHICS 0 to the screen.

```
10 GRAPHICS 0
20 DLIST=PEEK(560)+256*PEEK(561)
30 FOR I=0 TO 40
40 PRINT PEEK(DLIST+I);" ";
50 NEXT I
60 GOTO 60
```

If you choose to print the list to the line printer then change line 40 to;

```
40 LPRINT PEEK(DLIST+I)
```

It is quite useless to attempt to get an 80-column printer to print the display list across several rows instead of in a vertical column. Putting in a semi-colon at the end of the print line does little more than place two values on one line and this becomes confusing. The fault lies in the Operating System's printer driver.

The 32 byte long display list appears as follows for a 40K or larger computer with BASIC present;

```
112     Blank 8 scan lines to provide for overscan
112
112
 66 )   Display ANTIC mode 2 (BASIC 0) 64+2
 64 }   Screen memory starts at
156 )   64+156*256 = 40000
  2 \   Display ANTIC mode 2 for second mode line
  2 |
  2 |
  2 |
  2 |
  2 |
  2 |
  2 |
  2 |
  2 |   22 more ANTIC mode 2 modes
  2 |
  2 }
  2 |
  2 |
  2 |
  2 |
  2 |
  2 |
  2 |
  2 |
  2 |
  2 |
  2 /
 65 )   JVB—Jump and wait for Vertical Blank
 32 }   to display list address which starts at
156 )   32+256*156 = 39968
```

The first thing you notice is that every display list begins with three "blank 8 lines" instructions. This is to defeat the television's overscan by starting the display twenty-four scan lines down. The next instruction is a load memory scan (LMS). The ANTIC mode number is added to 64. The next two bytes are the low, high byte address to the beginning of display memory. Since the LMS instruction counts as the first display mode, only twenty-three more display ANTIC mode 2 instructions are needed. Finally there is a JVB instruction that resets the program counter to the top of the display list at loacation 39968.

The display list for a 16K machine is similar. The differences are in the locations of the top of the display list and the start of screen memory. The start of screen memory is at 15424 decimal. The low byte after the LMS instruction is 64 and the high byte is 60. The top of display list is at 15392 decimal. The low byte after the JVB instruction is 32 and the high byte is 60.

Mixing Graphics Modes

You are not always stuck with a homogenous stack of display modes. After all, splitting text and graphics is mixing modes. It is very easy to mix display modes just by changing a single display mode instruction in the display list. For example, we could change the twelfth row of GRAPHICS 0 text in the program below to a GRAPHICS 1 elongated text mode (ANTIC 6) characters by changing the 12th display instruction in the display list.

```
20 GRAPHICS 0
30 DLIST=PEEK(560)+PEEK(561)*256
40 FOR I=1 TO 24
50 PRINT "LINE";I;" THIS ROW HAS FORTY CHARACTERS";:IF I<24 THEN ?
60 NEXT I
1000 GOTO 1000
```

If we count down the display list starting with the 0th byte, a POKE DLIST+16,6 would change the text characters to GRAPHICS 1 characters. Do this by adding; 70 POKE DLIST + 16,6. The trouble is that everything below our new mode line is offset by half a row or twenty bytes. To understand why this occurs you need to understand what happens when ANTIC receives instructions to display a particular mode.

When ANTIC gets an instruction to display a particular graphics or character mode it automatically goes to display memory and gets the precise number of bytes of data necessary to display that mode line. When it sees an ANTIC 2 (GRAPHICS 0) display mode it retrieves forty bytes and interprets them in the proper display mode. When it sees another ANTIC 2 display mode it retrieves the next forty bytes in sequence from display memory. If instead it sees an ANTIC 6 (GRAPHICS 1) display mode, it only retrieves twenty bytes. Upon encountering the next ANTIC 2 display mode ANTIC retrieves another forty bytes. The trouble is that each of our print statements start at intervals of exactly forty bytes from the beginning of screen memory. When ANTIC only retrieved twenty bytes, it displayed only half of our printed line. The next ANTIC 2 display mode retrieves memory beginning with the last half of our line of text and displays it on the next line. Each of the ANTIC mode 2 lines on subsequent lines are also off by twenty bytes. You could correct this by adding another ANTIC 6 mode instruction immediately below the first one. Add line 80 POKE DLIST+17,6 to the program. The text for the twelfth row is now split between the two display lines of elongated text. Since we retrieved forty bytes less memory in producing the screen, the last row in memory isn't displayed.

You can experiment by changing any display instruction to any desired mode. If you try substituting BASIC GRAPHICS 2 (ANTIC 7) characters for those same two lines by POKEing a 7 instead;

```
70 POKE DLIST+16,7
80 POKE DLIST+17,7
```

the bottom of the display gets pushed downward, partially off screen.

What has happened is that we are now displaying more than 192 scan lines, 208 lines to be exact. While this isn't a major problem, it is possible to confuse the television screen's timing and the picture may begin to roll. As a rule of thumb, you shouldn't have more than 192 scan lines in your display. Displaying fewer scan lines will cause no problems. In fact it will decrease the 6502 execution time by reducing the number of cycles stolen by ANTIC to display the screen data.

Moving The Text Window

In BASIC, whenever you specify mixed text in any graphics mode, the four lines of text are automatically placed at the bottom of the screen. This is rarely the best location for it. It is often preferable when designing games to put your scoring information at the top. The text window is easy to move if you rewrite the display list.

If you look at a GRAPHICS 3 display list you will obtain the following values. The display list on the left is the one BASIC defaults to, and the one on the right is the display list we need to have four lines of text at the top and a GRAPHICS 3 screen below.

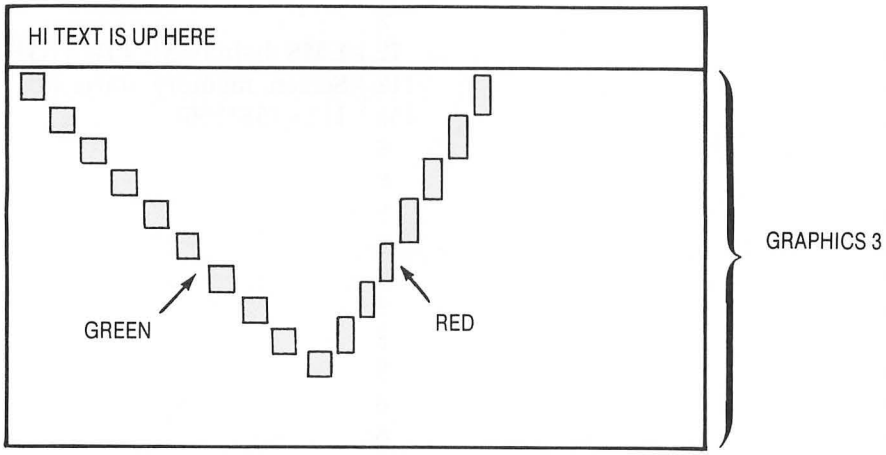
```
112  Blank 8 lines
112
112
 72 ) LMS display ANTIC 8
112 } Screen memory starts at
158 ) 112+(158*256)
  8  Display ANTIC 2 (GR.0)
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
 66 ) LMS display ANTIC 2
 96 ) Text memory starts at
```

```
112  Blank 8 lines
112  to provide for overscan
112
 66 ) LMS display ANTIC 2 (GR.0)
 96 ) Text memory starts at
159 )
      96+(159*256)
  2
  2
  2
 72 ) LMS display ANTIC 8 (GR.3)
112 } Screen memory starts at
158 ) 112+(158*256)
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
```

159	96+(159*256)	8	
2	Display ANTIC 2	8	
2		8	
2		8	
65	JVB	65	JVB
78	Address to top of DLIST	78	Address to top of DLIST
158	78+(158*256)	158	78+)158*256)

There are two ways to modify the display list. The first method changes only the groups of bytes that need to be modified. If the change is simple, you need rewrite only a fraction of an entire display list. In this example we are only shifting sections of the display list in order to change the position of the text window, so we can move blocks of display list data around if we are careful not to overwrite data. The diagram shows the sequence of the three moves. The actual move is accomplished in three short FOR-NEXT loops. The three data bytes at DLIST+3, 4, and 5 are read and POKEd into locations DLIST+9, 10, and 11. The other two moves are similar.

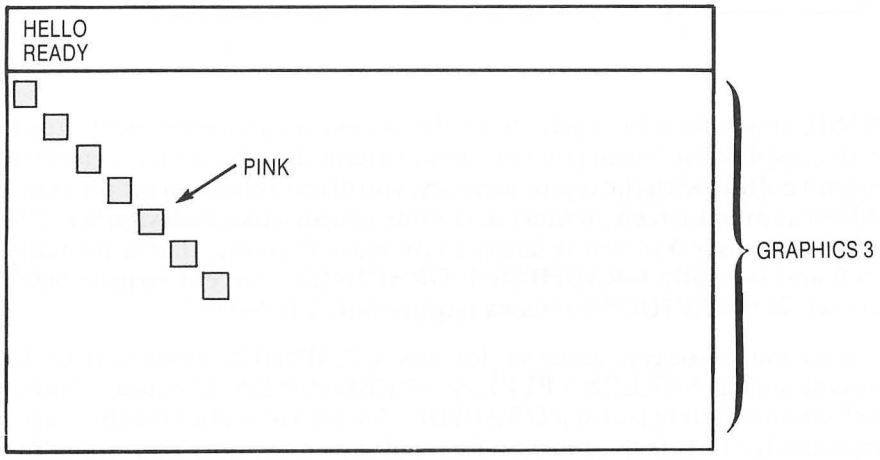
```
20 GRAPHICS 3
30 DLIST=PEEK(560)+PEEK(561)*256
35 REM MODIFY DISPLAY LIST
40 FOR I=0 TO 2:POKE DLIST+9+I,PEEK(DLIST+3+I):NEXT I
50 FOR I=0 TO 5:POKE DLIST+3+I,PEEK(DLIST+25+I):NEXT I
60 FOR I=0 TO 5:POKE DLIST+25+I,8:NEXT I
70 REM PLOT GR. 3
80 PRINT "HI TEXT IS UP HERE"
90 SETCOLOR 0,12,4:REM SET COLOR 1 TO GREEN
100 SETCOLOR 2,4,6:REM SET COLOR 3 TO PINK
110 COLOR 1:PLOT 0,0:DRAWTO 10,10
120 COLOR 3:PLOT 12,12:DRAWTO 19,0
1000 GOTO 1000
```



BASIC doesn't even notice the change. It keeps internal pointers to the locations of both text memory and screen memory. Since we didn't change these values text is printed correctly to the text area, and graphics are plotted correctly on the shifted screen. You might observe that when we set the colors for plotting in GRAPHICS 3, we affected the background of the text window. This is because SETCOLOR 2 affects the text background. We avoided plotting with SETCOLOR 1 because this register affects the luminance of the letters. We won't have any problem if we set this color register as long as we keep in mind that the luminance must be 6 or greater for readablity.

The second method and sometimes the easier method when rewriting the display list is to put the new list in data statements. Then it is only a matter of reading the list and POKEing the values into memory starting at the top of the old display list. Since the screen does act funny during the change, ANTIC can be temporarily disabled by writing a zero into SDMCTL at location 559 ($22F). After the list has been POKEd into position, you turn ANTIC back on with a POKE 559,34. If you did it right, the screen will appear exactly as you set it up. If for some strange reason you decide to move the display list, remember to tell the operating system by POKEing the address to the top of the list in locations 560 and 561 ($230,231), low and high bytes respectively.

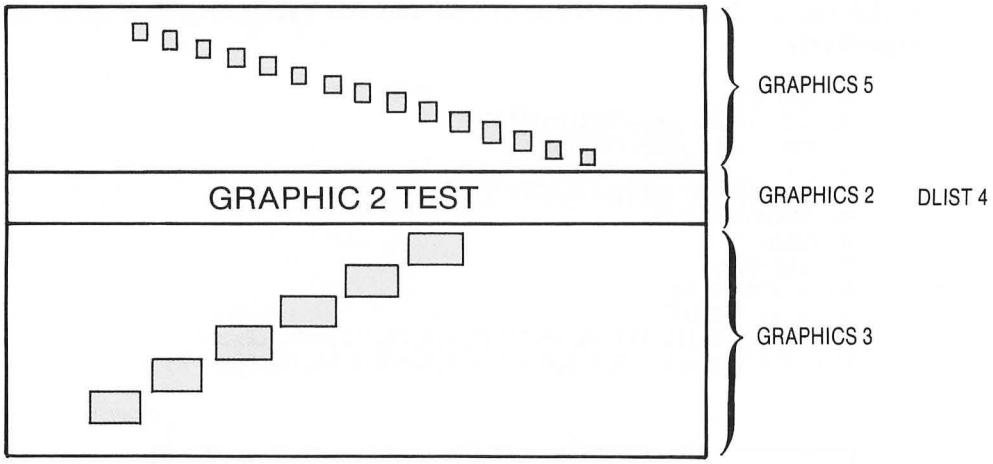
```
20 GRAPHICS 3
30 DLIST=PEEK(560)+PEEK(561)*256
35 REM MODIFY DISPLAY LIST
40 FOR I=0 TO 33
50 READ A:POKE DLIST+I,A:NEXT I
55 SETCOLOR 1,5,8
60 COLOR 2
70 PLOT 2,2
80 DRAWTO 10,10
90 PRINT "HELLO"
100 DATA 112,112,112,66,96,159,2,2,2,72,112,158,8,8
110 DATA 8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,65,78,158
```



Custom Display List For Mixing Graphics Modes

The last example is a custom designed display list with two graphics modes split by a row of expanded text. Whenever you decide to design any custom screen it is best to lay out the design on paper, and translate it into a sequence of mode lines. Once you have looked up the number of scan lines required for each mode line, you must double check the line count so that it does not exceed 192 scan lines. You then translate the sequence of mode lines into a sequence of ANTIC mode bytes.

Our display will consist of sixteen mode lines or rows of GRAPHICS 5 (ANTIC 10) pixels followed by a row of enlarged GRAPHICS 2 (ANTIC 7) text, followed by fourteen mode lines or rows of GRAPHICS 3 (ANTIC 8) pixels. Since each of the GRAPHICS 5 pixels are four scan lines high, this portion of the screen requires 16 x 4 = 64 scan lines. The one line of GRAPHICS 2 text requires sixteen scan lines. GRAPHICS 3 mode lines are eight scan lines high. The fourteen rows at the bottom require 14 x 8 = 112 scan lines. This gives us a total of 192 scan lines.



In BASIC it is sometimes easier to let the system set up the memory area for your screen and display list. Since you need to be careful that the screen memory requirements don't collide with the top of memory, you always choose a graphics mode that uses at least as much screen memory as the one you are custom designing. Obviously a GRAPHICS mode 5 screen is larger in memory than one that is partially GRAPHICS 5 and partially GRAPHICS 3. GRAPHICS 5 screens require 960 bytes of memory while GRAPHICS 3 screens require only 240 bytes.

The location of screen memory for any GRAPHICS mode can be found at locations 88 and 89. SCREEN = PEEK(88)+PEEK(89)*256. The individual values of the low byte and high bytes for a GRAPHICS 5 screen in a 40K+ machine are 160 and 155 respectively. This is the lowest address of screen memory corresponding to the upper left corner of the screen. Memory builds upwards towards the top of memory.

Each of our sixteen mode lines of GRAPHICS 5 pixels requires twenty bytes. Therefore that portion of the screen requires 16 x 20 = 320 bytes. If we add 320 bytes to the beginning of screen memory, the next byte (the beginning of our text memory) in low byte/high byte order is at:

	High Byte	Low Byte	
	155	160	
+		320	
	155	480	reduce to a value (0-255)
or	156	224	

The reason for the apparently nonsensical decimal math is that when you complete the addition of 320 and 160 in the low order byte the resulting decimal value 480 is larger than the storage capacity of one byte (255 decimal or 11111111 binary). The carry bit is set and the high order bit of the low byte is carried to the low order bit of the high byte. The resulting address is the beginning of text memory for our GRAPHICS 2 text. Since it requires only twenty bytes of display memory, the beginning of our GRAPHICS 3 memory is at 244 for the low byte and 156 for our high byte. With this information it is now possible for us to create a display list. That display list is shown below.

```
112     Blank 8 lines
112
112
 74 )   LMS Display ANTIC mode 10 (GR.5)
160 }   Screen memory starts at
155 )   160+(155*256)
 10     Display ANTIC mode 10
 10
 10
 10
 10
 10
 10
 10
 10
 10
 10
 10
 10
 10
 10
 10
 71 )   LMS Display ANTIC mode 7 (GR.2)
```

```
224 }  Text memory starts at
156 }  224+(156*256)
 72 }  LMS Display ANTIC mode 8 (GR.3)
244 }  Screen memory starts at
156 }  244+(156*256)
  8    Display ANTIC mode 8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
  8
 65 }  JVC (Jump and Wait for VBLANK)
104 }   Address of top of display list
155 }  104+(155*256)
```

Great care should be taken when designing a display list. One of the most frequent sources for error is forgetting that the LMS instruction is your first mode line for a particular graphics type. Forgetting this will cause you to have too many scan lines. The second source of error is incorrectly calculating the address for display memory. Once you begin looking at the wrong portion of memory, nothing appears that you print, plot, or POKE to.

Plotting Points & Lines Using A Custom Design

Plotting GRAPHICS 5 pixels on the top portion of our display is straightforward. All that you need to do is choose a color register, then PLOT. But on the lower portions of the screen, the Operating System must first be told which graphics mode to plot in and where screen memory is located. The current display mode is stored in memory location 87. Many programmers use this location to fool the OS into thinking that it is in a different GRAPHICS mode by POKEing it with a number from 0 to 11. This value is the same as the BASIC graphics mode number. In addition, the lowest address of screen memory stored at locations 88 and 89 must be changed.

In our example, when we wanted to plot to the GRAPHICS 3 portion of the screen, we POKED a 3 into location 87. We then put the beginning of the screen address into locations 88 and 89. These are the same values as our LMS address for

this portion of the screen. If you don't perform this operation the OS will incorrectly calculate the memory addresses to plot your pixel or line of pixels. Similarly the OS needs to be informed when we print our message in GRAPHICS mode 2 characters. A POSITION statement must also be included because the OS automatically does a position each time it plots. The #6 type print statement must be used because this is screen memory, not text memory.

In summary, the ability to design a custom display list is very valuable. It gives you flexiblity that is not available on other computer systems. By customizing the display you can give the screen a personality all your own.

```
20 GRAPHICS 5+16
30 DLIST=PEEK(560)+PEEK(561)*256
35 REM MODIFY DISPLAY LIST
40 FOR I=0 TO 42
50 READ A:POKE DLIST+I,A:NEXT I
100 COLOR 3:PLOT 10,0:DRAWTO 60,15
110 REM
120 POKE 88,244:POKE 89,156:POKE 87,3
130 COLOR 2:PLOT 19,0:DRAWTO 0,8
140 POKE 88,224:POKE 89,156:POKE 87,2
145 REM POSITION STATEMENT NEEDED FOR THE O.S. TO KNOW WHERE TO PUT CHARACTERS
150 POSITION 0,0
160 REM THE O.S. AUTOMATICALLY DOES A "POSITION" EACH TIME A PLOT IS DONE
170 ? #6;"  GRAPHICS 2 TEST"
500 DATA 112,112,112,74,160,155
510 DATA 10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
520 DATA 71,224,156
530 DATA 72,244,156
540 DATA 8,8,8,8,8,8,8,8,8,8,8,8,8
550 DATA 65,104,155
1000 GOTO 1000
5100 DATA 112,112,112,74,160,155
```

CHAPTER 3

CHARACTER SET GRAPHICS

The Atari computer is one of the few machines endowed with enormous flexibility when it comes to handling character set graphics. Most computers have a set of character shapes stored permanently in Read Only Memory (ROM). The Atari is no exception. In addition to the usual upper and lowercase letters and numbers, twenty-nine control keys represent graphics symbols. These include a heart, diamond, spade, and club useful for creating playing cards, and a number of triangles, small squares, and diagonal and edge lines suitable for creating geometric graphic displays. While special characters can be useful in some playfield designs, it is impossible to achieve widely varied playfield designs using characters on most computers without resorting to slower bit-mapped character shapes.

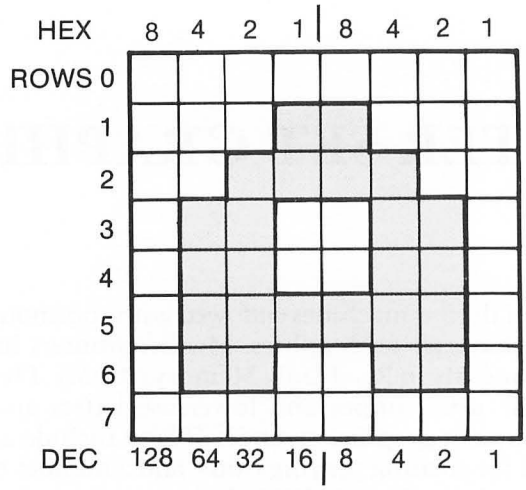
The Atari, fortunately, takes the concept of character graphics one step further and allows the user to redefine the character shapes to create colorful and varied backgrounds. A hardware register and its shadow register in RAM keeps track of the location of the character set currently being used. Normally the register defaults to the set stored in ROM at $E000, but you can give it any RAM address that falls on a 1K boundary. When ANTIC is given instructions to put character data on the screen it fetches character data from the character set stored at this location.

A character set consists of a maximum of 128 eight-byte shapes. Each character is represented on the screen as a group of 8 x 8 dots or pixels. When reading text on the screen it appears that there are gaps between characters, but there aren't. The gaps are provided by leaving blank space near the edge of the block alloted to the individual characters. Since there is no space otherwise defined between characters, it is possible to create larger shapes consisting of groups of adjacent characters, or background with no breaks in it.

If we take a look at a single character like the capital letter A from the ROM character set, it forms a pattern of on-off dots as illustrated below. The pixels in each of the eight rows, numbered here from 0 -7, are represented in the character set as a value from 0 -255 ($00-$FF). The positions of the individual pixels in the row determine the value.

Pixel values are calculated using a base two numbering system. If a pixel is in the rightmost column of a row, the bit is set, and it has a value of one. If the lit pixel is in the next column to the left, that bit is set, and it has a value of two. The values increase by a factor of two until we reach the left most column which has a value of 128. Thus, if you want to determine the value of the row in a character you add up the individual bit values for the lit pixels. If we look at row #1 in the letter A we find that the fourth and fifth pixels are lit. The value of the set bits is 8+16 = 24 ($18). If all the pixels were lit we would have 128+64+32+16+8+4+2+1 = 255 ($FF).

3 CHARACTER SET GRAPHICS



HEX	DEC
$ 0.0	00
$ 1 8	24
$ 3 C	60
$ 6 6	102
$ 6 6	102
$ 7 E	126
$ 6 6	102
$ 0 0	00

Each character is stored sequentially in the character set as groups of eight values. The first character is stored in the first eight memory locations (0-7) in the character set, the second character is stored in the second eight memory locations (8-15), etc. The first character is called the 0 th internal character, the second the first, etc. Eight values times 128 characters requires 1024 bytes of memory. If you look in your BASIC book you will see that internal character values 128-255 produce inverse characters. These characters are the same as the ones in the internal character set except the seventh or leftmost bit of the character number is set or turned on. This is equivalent to adding 128 to the internal character number. If the high bit is set, ANTIC automatically interpretes the character as inverse, and it plots it on the screen.

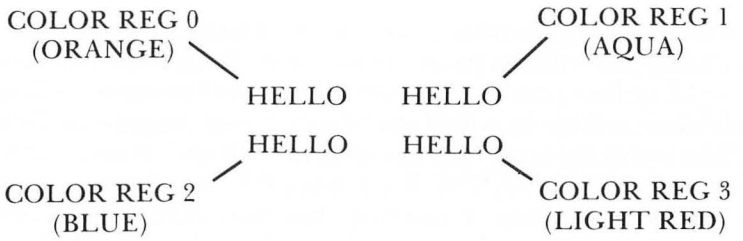
Since the computer can only store numbers from 0-255, all characters are assigned ATASCII (Atari ASCII) numbers. For example the letter A is assigned the ATASCII value 65. If you look at the individual characters stored internally from 0-127 in the character set and then compare them to the ATASCII values you will find them out of order. In fact the order is as follows:

TYPE	ATASCII ORDER	MEMORY ORDER
uppercase numbers, punctuation	32-95	0-63
graphics characters	0-31	64-95
lowercase, some graphics	96-127	96-127

Essentially, Atari moved the graphics characters and placed them between the uppercase and lowercase letters. It may seem illogical at first since it complicates the calculation of the memory locations for any ATASCII character value, but it actually enables you to choose between upper and lowercase graphics in modes one and two. In both of these enlarged text modes only sixty-four characters are available, and the set is only 512 bytes long. They use the leftmost two bits of each byte to point to the color register for that character. The uppercase characters in internal positions 0-63 use color register #0. If you attempt to print lowercase characters to the screen, you still get uppercase characters but using a different color register. The letter "A" is internal character 33 while the letter "a" is internal character 97. This is equivalent to 33 + 64 or toggling one of the two high bits. The same is true for inverse. This can be demonstrated in the following program.

```
10 GRAPHICS 2+16
20 PRINT #6;"HELLO hello"
30 PRINT #6;"HELLO hello"
100 GOTO 100
```

The result is the word HELLO printed in capital letters to the screen in four different colors. This occurs because the half-size, 512-byte character set doesn't contain any inverse or lowercase letters. The computer interprets only the lower six bits of the ASCII character as the character number and the upper two bits as the color register. Uppercase letters use color register 0 so they appear in orange. Lowercase characters are interpreted as color register 1 and appear in aqua. Inverse uppercase characters, which appear in the second line, use color register 2 and are blue, while inverse lowercase uses color register 3 and are light red. Remember that these colors are the computer's default values and the user can change them. The PRINT #6 prints to the graphics portion of the screen. Regular PRINT statements print to the text window, if there is one.



There is a method of obtaining lowercase letters on the screen, but if you use it you can't have uppercase characters, too. Since lowercase characters are in the upper 512 bytes of the character set, you can tell the computer that this is your new character set by changing the character set address pointer to an address that is 512 bytes higher. Location 756, the character base shadow register, normally points to the character set at $E000 and has a value for the high byte of 224 ($E0). If you add the line

```
40 POKE 756,226
```

the characters will be printed from this alternate set. They will all appear as lowercase in different colors. If you want to mix upper and lowercase letters you will need to move the character set into RAM and copy the lowercase characters into the locations occupied by the numbers and symbols.

Finding Your Character Within a Set

If you are going to move or change characters you have to be able to find them in memory by either internal or ATASCII value. Finding them by internal value is simple. Each character is 8 bytes long. The formula is;

MEMORY LOCATION = START+ 8 * INTERNAL VALUE

Finding them by ATASCII value is more difficult because the graphics characters have been sandwiched between the upper and lowercase letters. The three formulas are as follows. Please remember that the starting value of the character set defined by START should be on a 1K boundary. That value is 57344 if you are using the ROM character set.

ATASCII VALUE (AV)	STARTING MEMORY LOCATION
32-95	START+(AV-32)*8
0-31	START+(AV+64)*8
96-127	START+(AV*8)

Changing the Character Set

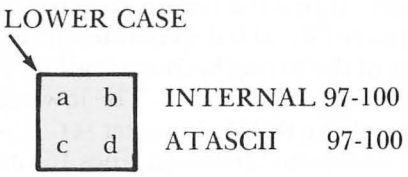
The easiest method of customizing a character set is to copy the ROM character set to RAM and change individual characters within it. To do this from BASIC you will need to reserve 1K or four pages of memory at the top of memory so that the set will reside in a safe place and not be wiped out by either your program or the display area of memory. The top of memory for any computer is found at decimal location 106. The actual value is PEEK(106)*256. If we move this high byte pointer down four pages there will be a new top of memory. We then POKE this value back into location 106 so that BASIC won't put anything above this. The RAM character set begins at this new top of memory. Do a graphics call after changing location 106.

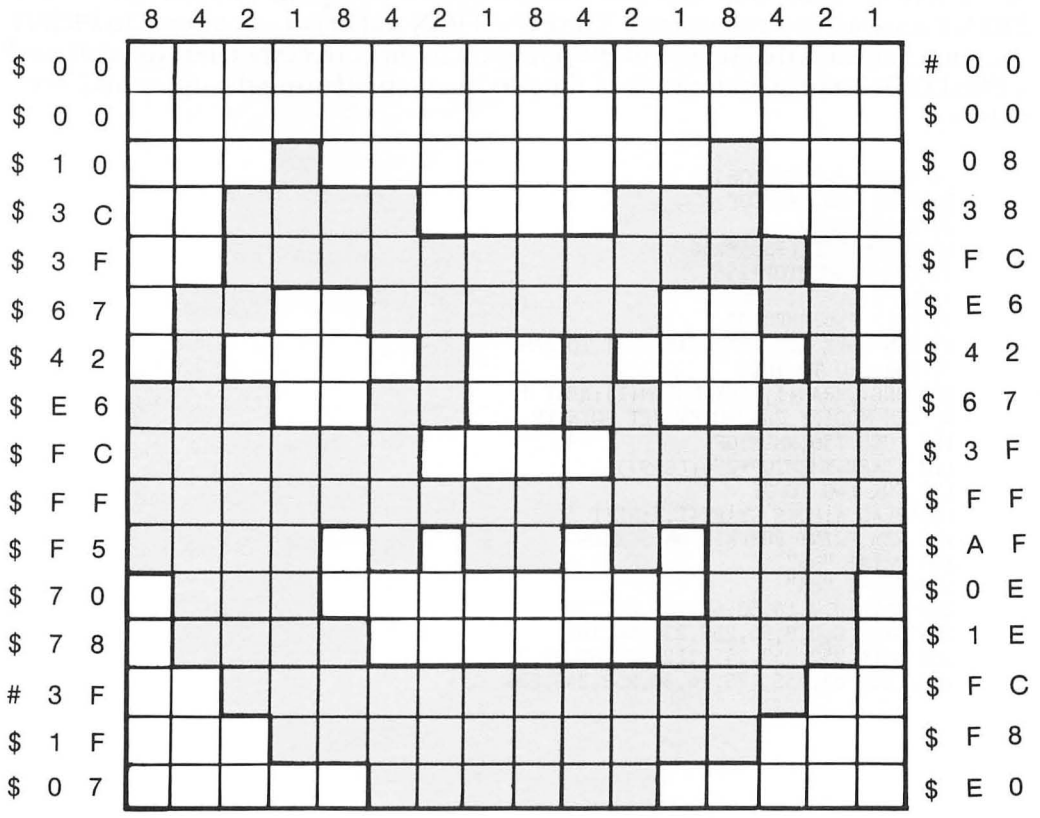
We must now inform ANTIC of the new location for the character set. Location 756 ($2F4) is the character base shadow register. The value in this location is copied into the character base hardware register at location 54281 ($D409) every sixtieth of a second during the vertical blank period. We can't store this value directly in the hardware address, or it will be overwritten during the next vertical blank period.

Moving the character set requires reading a byte from ROM and storing it in the appropriate position in RAM. In our case CHROM is 57344 ($E000) and CHRAM =

NMEMTOP*256. To accomplish this PEEK the values in ROM and POKE them into RAM during a FOR-NEXT loop from 0 to 1023. Once you have copied the set you can then modify specific characters and use these in PRINT statements to the screen. Or if you wish, you can alternately POKE their internal character numbers directly into screen memory. This will require a calculation to determine the correct position.

In order to show the power of a redefined character set, we will create a large Halloween pumpkin that consists of four adjacent lowercase characters, the letters a, b, c and d. The diagram below shows a magnified view of the pumpkin's lit pixels along with the screen arrangement of the four characters or letters that they represent. The letter "a" when redefined shows only the upper left quarter of our pumpkin. Similarly, the letter "b" shows only the upper right portion of our pumpkin.





Once the pixel positions in the eight rows of each character are translated into numerical data equivalent to our drawing, they are then POKEd into the appropriate position in the RAM character set as replacements to the existing characters. The four lowercase letters are internal characters 97-100. Since the four are next to each other, we can POKE in all four characters together in a FOR-NEXT loop. The starting location of the ninety-seventh character is the starting location of the RAM character set offset by 8 bytes x 97 characters. Therefore

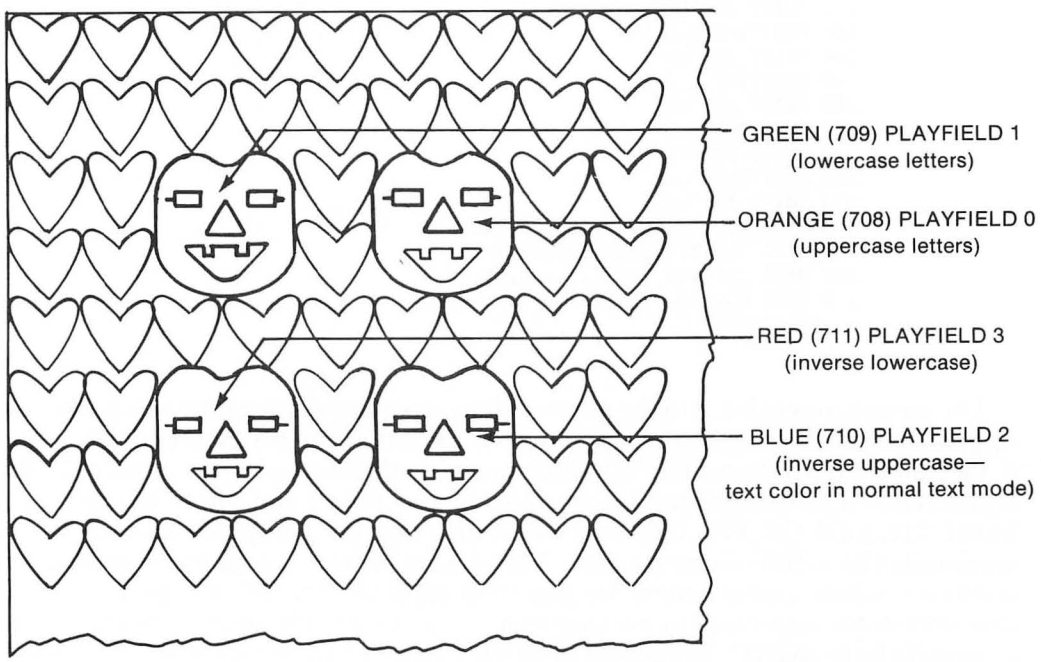
START = NMEMTOP*256 + (8*97)

Copying the character set into RAM using a FOR-NEXT loop is quite slow and requires ten seconds. Normally, you copy the set first and then change the character set pointer at decimal location 756. I thought it might be more fun to watch the set being copied so I listed the program to the screen and changed the character base pointer before the copy loop. At first the listing appears fine but when the character set points to a garbage section of RAM it degenerates into random dots. As the ROM set is copied more and more of the listing becomes legible. The numbers and symbols appear first, followed by the uppercase letters. The lowercase letters form last, since they are in the last quarter of the ROM character set. If you stare at the lowercase letters in our two PRINT statements listed on lines 160 and 170, you will see them change into a pumpkin. The pumpkin will then be printed in the upper left corner of the screen. Each time that you wish to draw a pumpkin you have to use two PRINT statements on the screen. The POSITION statement can be used to PRINT the four characters that represent the pumpkin in the correct spot, but you will need a POSITION statement for each of the two lines representing the upper and lower halves.

```
10 NMEMTOP=PEEK(106)-4
20 POKE 106,NMEMTOP
30 GRAPHICS 0
40 CHROM=PEEK(756)*256
50 CHRAM=NMEMTOP*256
55 LIST
56 POKE 756,NMEMTOP
60 REM COPY ROM CHARACTER SET TO RAM
70 FOR I=0 TO 1023
80 POKE CHRAM+I,PEEK(CHROM+I):NEXT I
90 REM MODIFY CHARACTER SET POINTER
100 POKE 756,NMEMTOP
110 START=NMEMTOP*256+(8*97)
130 FOR I=0 TO 31
140 READ A:POKE START+I,A:NEXT I
150 REM PRINT PUMPKIN ON SCREEN
160 PRINT "ab"
170 PRINT "cd"
200 DATA 0,0,16,60,63,103,66,230
210 DATA 0,0,8,56,252,230,66,103
220 DATA 252,255,245,112,120,63,31,7
230 DATA 63,255,175,14,30,252,248,224
```

Many games use character set playfields as backgrounds in either BASIC mode 2 (ANTIC 7) or ANTIC mode 4. The latter allows small but detailed four-color characters four pixels wide by eight deep but requires nearly 2K of screen memory. While individual characters are limited to a single color in BASIC mode 2, they have an advantage in that these large characters don't require much memory. With twelve rows of twenty characters a screen requires only 240 bytes of memory. Each character position can be plotted in one of four colors, determined by toggling the two high bits of the internal character number.

The next example is similar to the last, except that the redefined characters are plotted as BASIC mode 2 characters. Again, space is reserved above the top of memory by moving down the top of memory pointer four pages (1024 bytes). The ROM character set is copied as before, but only half of the set, or 512 bytes, is used in this graphics mode. The shortened set doesn't contain both upper and lowercase letters. If you use the lower half of the set, you get uppercase letters only, even if you print lowercase letters. To print the four redefined lower case letters you will need to change the character set pointer to the upper half of the RAM character set. This can be accomplished in line 100 by adding two pages to NMEMTOP and POKEing it into the character set pointer at location 756.



```
10 NMEMTOP=PEEK(106)-4
20 POKE 106,NMEMTOP
30 GRAPHICS 2
40 CHROM=PEEK(756)*256
50 CHRAM=NMEMTOP*256
55 LIST
56 POKE 756,NMEMTOP
60 REM COPY ROM CHARACTER SET TO RAM
70 FOR I=0 TO 1023
80 POKE CHRAM+I,PEEK(CHROM+I):NEXT I
90 REM MODIFY CHARACTER SET POINTER
95 REM GRAPHICS 2 CHARCTER SET ONLY 512 BYTES
96 REM AND LOWER CASE IS DISPLAYED AS UPPER CASE LETTERS
97 REM TO REACH UPPER HALF OF SET ADD 2 PAGES TO NMEMTOP
100 POKE 756,NMEMTOP+2
110 START=NMEMTOP*256+(8*97)
130 FOR I=0 TO 31
140 READ A:POKE START+I,A:NEXT I
150 REM PRINT PUMPKIN ON SCREEN
155 POSITION 2,2
160 PRINT #6;"ab"
165 POSITION 2,3
170 PRINT #6;"cd"
175 POSITION 5,2
180 PRINT #6;"AB"
185 POSITION 5,3
190 PRINT #6;"CD"
195 POSITION 2,5
200 PRINT #6;"ab"
205 POSITION 2,6
210 PRINT #6;"cd"
215 POSITION 5,5
220 PRINT #6;"AB"
225 POSITION 5,6
230 PRINT #6;"CD"
240 DATA 0,0,16,60,63,103,66,230
250 DATA 0,0,8,56,252,230,66,103
260 DATA 252,255,245,112,120,63,31,7
270 DATA 63,255,175,14,30,252,248,224
```

The pumpkin is printed to the screen in four separate positions. By using all of the uppercase and lowercase normal and inverse combinations for the letters a, b, c, and d, the pumpkin will appear in four different colors determined by the playfield color registers 0-3. The background is controlled by playfield 4 and can be changed by a POKE 712, COLOR VALUE. You will notice that the pumpkins are completely surrounded by yellow colored hearts. This occurs because screen memory contains zeros everywhere except where we placed pumpkins. ANTIC interpretes the 0th character in the lower half of the character set as a heart. The hearts can be removed from sight by POKING the color in playfield 0 to zero, but one of the pumpkins will also fade from sight. Therefore, it is best to create a blank character by POKEing zeros into this 0th character. To do this, add the following lines.

```
234 START=(NMEMTOP+2)*256
235 FOR I=0 TO 7
236 POKE START+I,0:NEXT I
```

Copying the ROM character set is the slowest part of the program. This operation could be speeded up a hundred or more times by substituting a Machine language subroutine that can be called by BASIC's USR function. In order to make the routine versatile, the user is given the option of either relocating the full 1024 byte ROM character set to RAM, or just the upper 512 bytes as might be needed in a Graphics 2 display. The routine is also fully relocatable and presently resides in the upper half of page six in memory. Its calling format is A= USR(1664,CHRAMH, 1 or 2). The number 1664 is the starting location of the Machine languge subroutine. CHRAMH is the high byte value of the relocated RAM character set. This should be on a 1K page boundary for full sized (1024 byte) character sets and on a ½K page boundary for half size (512 byte) character sets. The value one tells the routine to move all 1024 bytes of the ROM character set to the new RAM location, while a two tells the routine to skip the first 512 bytes of the ROM character set and just move the last half of the set to the new RAM location. The Graphics 0 mode pumpkin example is repeated below to show you the obvious speed advantage in using the Machine language subroutine.

```
10 NMEMTOP=PEEK(106)-4
20 POKE 106,NMEMTOP
30 GRAPHICS 0
40 LIST
50 POKE 756,NMEMTOP
60 REM READ IN CHARMOVE SUBROUTINE
70 FOR I=0 TO 49
80 READ A:POKE 1664+I,A:NEXT I
90 A=USR(1664,NMEMTOP,1,4)
100 REM MODIFY CHARACTERS a,b,c,d INTERNAL 97-100
110 START=NMEMTOP*256+(8*97)
120 FOR I=0 TO 31
130 READ A:POKE START+I,A:NEXT I
140 REM PRINT PUMPKIN ON SCREEN
150 PRINT "ab"
160 PRINT "cd"
890 REM DATA FOR MACHINE LANGUAGE MOVE ROUTINE
900 DATA 104,169,224,133,206,104,104,133,204,104,104,201,1,240,4,230
910 DATA 206,230,204,104,104,141,176,6,169,0,133,205
920 DATA 133,203,168,177,205,145,203,200,208,249,230
930 DATA 206,230,204,206,176,6,208,240,96,0,0
990 REM DATA FOR 4 PUMPKIN CHARACTERS
1000 DATA 0,0,16,60,63,103,66,230
1010 DATA 0,0,8,56,252,230,66,103
1020 DATA 252,255,245,112,120,63,31,7
1030 DATA 63,255,175,14,30,252,248,224
```

Note: Assembly language listing of subroutine in appendix

Character Editor

Customizing a character set can be rather tedious, if you don't use a character set editor. We have furnished a simple one that you can operate by either keyboard or joystick control. The editor allows you to design single-color characters and afterwards save your custom set to a disk for later use in your own program.

The screen displays an enlarged 8 x 8 grid at the top for editing or designing your

characters. The entire 128 character set is displayed below. Newly edited characters are also displayed in their proper position within the set. A character is chosen for editing by giving the internal character number. You should use the appendix in the back of this book to choose the correct character. The CTRL CLEAR key will erase the bit pattern of the present character if you wish to design rather than modify an old character. A white square above the enlarged grid indicates that you are in the Draw mode. It can be turned off and will erase by pressing the U key for Undraw. It can be toggled back by pressing the D key. You can move the cursor without affecting anything by moving the joystick or by using the arrow keys. Each time that you want to draw a pixel you can either press the joystick button or the space bar. Since pressing the space bar for each pixel is tedious, I recommend that you edit using a joystick. When you are finished editing the character, just hit return to enter it. You can avoid entering a newly edited character by pressing the ESC key.

Once you are completely satisfied with your custom set, you can save it to the disk by pressing the S key for Save. You only need give it a file name, and it will be saved to disk. If you are re-editing a previous set, you can load one from disk using the L key for Load. You don't have to worry about loading a set to edit the first time you use the utility, because the program automatically defaults to the ROM character set which it has copied into RAM.

```
5 REM CHARACTER SET EDITOR - BY DAN PINAL
10 REM HIT RESET BEFORE RERUNNING
20 REM SO MEMTOP IS RESET
30 POKE 106,PEEK(106)-16:GRAPHICS 0:REM RESERVE ROOM FOR 2 CHR SETS & PMG
40 PMBASE=PEEK(106):DLIST=PEEK(560)+256*PEEK(561):CB1=PEEK(106)+8:CB2=PEEK(106)+12
50 SET1=CB1*256:SET2=CB2*256
60 GOSUB 1000
70 POKE 559,62:POKE 54279,PMBASE:POKE 53277,3
80 POKE 703,4:POKE 752,1
90 DRAW=1
100 REM COMMAND
110 ? "SAVE,LOAD, OR EDIT"
120 GET #1,X
130 IF X=69 THEN 300:REM EDIT
140 IF X=76 THEN C$="L":CMD=4:GOTO 170
150 IF X=83 THEN C$="S":CMD=8:GOTO 170
160 GOTO 120
170 ? "NAME OF FILE";:INPUT Q$
180 F$="D:":IF Q$="" THEN 170
190 IF Q$(1,1)=":" OR Q$(2,2)=":" THEN F$=Q$:GOTO 210
200 F$(3)=Q$:REM OTHERWISE DEFAULT TO DRIVE 1
210 TRAP 250:IOCB=2:OPEN #IOCB,CMD,0,F$
220 X=USR(CIO,IOCB,SET1,1024,ADR(C$))
240 CLOSE #IOCB:GOTO 100
250 CLOSE #IOCB:? "I/O ERROR":GOTO 100
300 REM
310 TRAP 100:? "CHARACTER # TO EDIT OR RETURN";:INPUT CH:TRAP 40000
320 IF CH<0 OR CH>127 THEN 300
330 FOR L1=0 TO 7
340 TEMP=PEEK(SET1+CH*8+L1)
350 FOR L2=0 TO 7
360 X=L1*8+L2+1
370 B$(X,X)=CHR$(33):REM ASSUME BLANK (!)
380 IF TEMP>=BIT(L2) THEN B$(X,X)=CHR$(34):TEMP=TEMP-BIT(L2):REM (")
390 NEXT L2
400 POSITION 2,2+L1:? #6;B$(L1*8+1,L1*8+8);
```

```
410 NEXT L1
420 H=0:V=0
430 ? " ↑ ↓ ← →, SPACE BAR TO DRAW OR JOYSTICK"
440 ? "DRAW,UNDRAW, CTL CLEAR TO ERASE"
450 ? "ESC TO EXIT"
500 X=USR(PCURSOR,H,PLAYER0+V):S=S(STICK(0)):IF S>0 THEN POKE 764,S
505 IF  NOT STRIG(0) THEN POKE 764,33
510 IF PEEK(764)=255 THEN 500
520 GET #1,X
530 IF X<>45 THEN 550:REM UP
540 V=V-(V-1>=0):GOTO 500
550 IF X<>61 THEN 570:REM DOWN
560 V=V+(V+1<8):GOTO 500
570 IF X<>43 THEN 590:REM LEFT
580 H=H-(H-1>=0):GOTO 500
590 IF X<>42 THEN 610:REM RIGHT
600 H=H+(H+1<8):GOTO 500
610 IF X<>28 THEN 630:REM CTL UP
620 V=V-(V-1>=0):H=H+(H+1<8):GOTO 500
630 IF X<>29 THEN 650:REM CTL DOWN
640 V=V+(V+1<8):H=H-(H-1>=0):GOTO 500
650 IF X<>30 THEN 670:REM CTL LEFT
660 V=V-(V-1>=0):H=H-(H-1>=0):GOTO 500
670 IF X<>31 THEN 690:REM CTL RIGHT
680 V=V+(V+1<8):H=H+(H+1<8):GOTO 500
690 IF X<>32 THEN 710:REM SPACE
700 B$(V*8+H+1,V*8+H+1)=CHR$(33+DRAW):POSITION 2,2+V:? #6;B$(V*8+1,V*8+8);
701 BYTE=0:FOR L1=0 TO 7:X=V*8+L1+1
702 IF ASC(B$(X,X))-33 THEN BYTE=BYTE+BIT(L1)
703 NEXT L1:POKE SET1+CH*8+V,BYTE:GOTO 500
710 IF X<>68 THEN 730:REM D
720 DRAW=1:POKE SCREEN+2,128:GOTO 500
730 IF X<>85 THEN 750:REM U
740 DRAW=0:POKE SCREEN+2,0:GOTO 500
750 IF X<>125 THEN 770:REM CLEAR
760 FOR L1=0 TO 7:POKE SET1+CH*8+L1,0:NEXT L1
765 GOTO 330
770 IF X<>27 THEN 500:REM ESC
998 GOTO 100
999 STOP
1000 REM DLI& SETUP ARE 75 BYTES STARTING AT 1536
1010 DATA 104,169,6,162,6,160,11,32,92,228,96,169,48,141,0,2,169,6,141,1,2,169,192,141,14
1020 DATA 212,174,254,6,165,20,41,7,208,10,165,20,41,15,240,1,138,141,0,208,76,95,
          228,72,173
1030 DATA 255,6,141,9,212,169,67,141,0,2,169,6,141,1,2,104,64,72,169,224,141,9,212,104,64
1040 REM THIS CIO SUB. IS 49 BYTES
1050 DATA 104,104,104,10,10,10,10,170,104,157,69,3,104,157,68,3,104,157,73,3,104,
          157,72,3,104
1060 DATA 133,213,104,133,212,160,0,177,212,160,7,201,83,208,2,160,11,152,157,66,3,
          76,86,228
1070 REM OPEN #IOCB,IO,0,FILE$:X=USR(CIO,IOCB,BUFADR,BUFLEN,ADR("S" or"L"))
1080 REM SET MOVE IS 34 BYTES
1090 DATA 104,104,133,212,104,133,213,169,0,133,206,169,224,133,207,162,4,160,0,
          177,206,145
1100 DATA 212,200,208,249,230,207,230,213,202,208,242,96
1110 REM PCURSOR 41 BYTES
1120 DATA 104,104,104,10,10,105,56,141,254,6,104,133,213,169,0,133,212,168,145,212,
          200,208,251
1130 DATA 104,10,10,10,105,48,133,212,160,7,169,240,145,212,136,16,251,96
1132 REM CLEAR PM AREA SUBROUTINE
1134 DATA 104,104,133,213,104,133,212,162,0,160,0,169,0,145,212
```

```
1136 DATA 200,208,251,230,213,232,224,8,144,240,96
1140 SETUP=1536:CIO=SETUP+75:SETMOVE=CIO+49:PCURSOR=SETMOVE+34:CLEAR=PCURSOR+41
1150 FOR L1=1536 TO 1760:READ X:POKE L1,X:NEXT L1
1160 POKE 756,CB2:POKE 1791,CB1
1170 X=USR(SETMOVE,CB1):X=USR(SETMOVE,CB2)
1180 FOR L1=SET2 TO SET2+23
1190 READ X:POKE L1,X:NEXT L1
1200 POKE DLIST+14,130:POKE DLIST+24,130
1210 REM
1220 X=USR(SETUP)
1230 REM
1240 DATA 0,0,0,0,0,0,0,0
1250 DATA 255,129,129,129,129,129,129,255
1260 DATA 255,129,189,189,189,189,129,255
1270 POKE 703,4:POKE 752,1
1280 COLOR 33:FOR L1=2 TO 9
1290 PLOT L1,2:DRAWTO L1,9
1300 NEXT L1
1310 SCREEN=PEEK(DLIST+4)+256*PEEK(DLIST+5)
1320 ADDR=SCREEN+40*11+2
1330 FOR L1=0 TO 7:FOR L2=0 TO 31
1340 POKE ADDR,L1*32+L2:ADDR=ADDR+1
1350 NEXT L2:ADDR=ADDR+8:NEXT L1
1360 OPEN #1,4,0,"K:"
1370 DIM F$(16),Q$(16),C$(1),B$(64),BIT(7),BYTE(7),S(15)
1380 B$="!":B$(64)="!":B$(2)=B$
1390 BIT(0)=128:BIT(1)=64:BIT(2)=32:BIT(3)=16:BIT(4)=8:BIT(5)=4:BIT(6)=2:BIT(7)=1
1400 FOR L1=0 TO 15:S(L1)=0:NEXT L1
1410 S(5)=135:S(6)=142:S(7)=7
1420 S(9)=143:S(10)=(134):S(11)=6
1430 S(13)=15:S(14)=14
1440 POKE 704,10:POKE 705,70:POKE 706,70:POKE 707,70
1445 X=USR(CLEAR,PMBASE*256):REM CLEAR PM AREA
1450 PLAYER0=PMBASE*256+1024:PLAYER1=PLAYER0+256:PLAYER2=PLAYER1+256:PLAYER3=PLAYER2+25(
1460 FOR L1=44 TO 115
1470 POKE PLAYER2+L1,15:POKE PLAYER3+L1,15:NEXT L1
1480 FOR L1=0 TO 3:POKE PLAYER1+44+L1,255:POKE PLAYER1+112+L1,255:NEXT L1
1490 POKE 53257,3:POKE 53249,56:POKE 53250,48:POKE 53251,84
1500 RETURN
```

Character Set Loader

In order to use these custom character sets you will need a Machine language routine to load your custom character set into memory from disk. The subroutine can be accessed through a USR call. The format is U=USR(CALL,IOCB, SET,LENGTH,CMD). If you are placing it into page six, CALL = 1536. IOCB = 1 for a read. The device # that was opened is for the file (like open #1, etc.). The set is placed at SET=CB*256 where CB is the high byte location of the character set, and LENGTH =1024. The format CMD=ADR("L") is somewhat unusual. The subroutine was designd to accept either an "L" or "S" letter command. The statement physically passes the address where the letter is and the computer looks and finds the actual letter stored at that address.

The program asks for the name of the file. The "D:" doesn't need to precede the name because this string is included at the beginning of F$. The input name string

is Q$. The statement F$(3)=Q$ tacks the name of the file to the "D:". The one last thing that you have to do before you call the subroutine is to open the channel to access the drive. This is in the form OPEN #IOCB, IO,0,F$. IOCB=1 , IO=4, and the 0 is unused. I won't go into details of the actual Machine language routine. The listing, however, is provided for Machine language programmers; my comments accompany it.

```
10 REM CIO CALL TO LOAD DATA FROM ANY FILE - DAN PINAL
20 POKE 106,PEEK(106)-8:GRAPHICS 0:REM MOVE RAMTOP DOWN 2K
30 DIM Q$(14),F$(14):F$="D:":REM ASSUME A DISK LOAD
40 CB=PEEK(106)+4:SET=CB*256:CALL=1536
45 REM POKE IN MACHINE LANGUAGE ROUTINE INTO PAGE 6
50 FOR L1=0 TO 48:READ X:POKE 1536+L1,X:NEXT L1
60 ? "NAME OF FILE TO LOAD";:INPUT Q$:F$(3)=Q$
65 REM SET PARAMETERS FOR OPEN ROUTINE AND USR CALL
70 IOCB=1:IO=4:CMD=ADR("L"):LENGTH=1024
80 OPEN #IOCB,IO,0,F$
90 U=USR(CALL,IOCB,SET,LENGTH,CMD)
100 REM THAT'S IT. POKE 756,CB TO TURN ON YOUR SET
110 FOR L1=0 TO 255:? CHR$(27);CHR$(L1);:NEXT L1:POKE 756,CB
10000 DATA 104,104,104,10,10,10,10,170,104,157,69,3,104,157,68
10005 DATA 3,104,157,73,3,104,157,72,3,104
10010 DATA 133,213,104,133,212,160,0,177,212,160,7,201,83,208
10015 DATA 2,160,11,152,157,66,3,76,86,228
10020 REM OPEN #IOCB,IO,0,FILE$:X=USR(CIO,IOCB,BUFADR,BUFLEN,ADR("S" or"L"))
```

Note: Assembly language listing in appendix

Multi-Color Characters

There are two important character graphics modes that are not supported directly by OS. Both ANTIC modes 4 and 5 use characters that are only four pixels wide instead of the usual eight pixels. Each byte is broken up into four bit pairs. The advantage is that each pixel can be of four different colors, counting the background color. Since each pixel is addressed by two bits instead of one, the value of each bit pair can determine from which color register the pixel derives its color. This becomes a very clever use of color indirection. Note that if the high bit of the character number is set, you get a fifth playfield color. The bit pair order and its associated playfield color register are as follows:

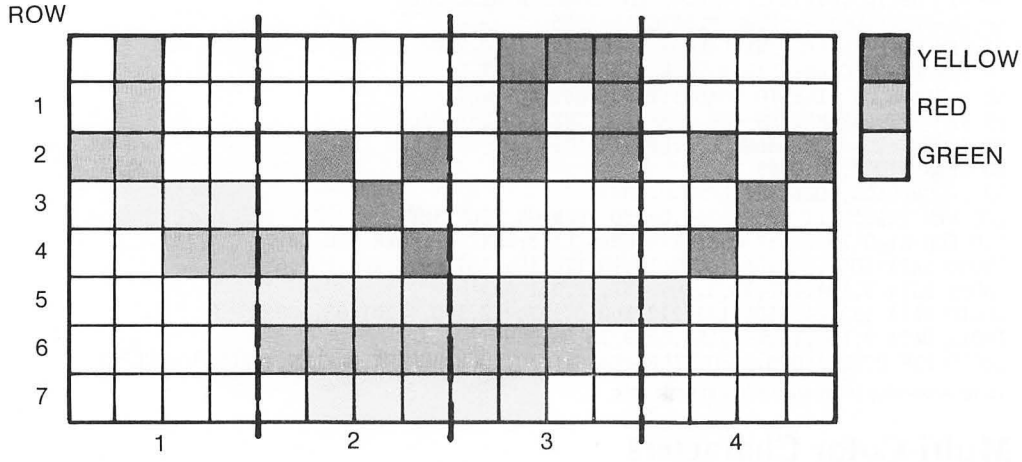
BIT PAIR		PLAYFIELD	COLOR REGISTER
0 0		4 (Bkd)	712
0 1		0	708
1 0		1	709
1 1	character #	2	710
1 1	high bit set	3	711

Multi-colored characters in ANTIC mode 4 can be used to create colorful and detailed playfields. They are also useful in games in which a large number of multicolored animated objects are required. While it is always tempting to use player-missile animation because it is easier, large detailed multicolored players aren't always available unless you overlap or combine your relatively few available players.

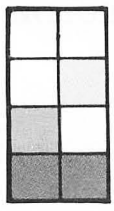
3 CHARACTER SET GRAPHICS

Most multi-colored ANTIC 4 shapes require a number of adjacent redefined characters since 4 x 8 pixel sized characters are tiny and don't allow for much detail. This group of characters is called a matrix.

To give you an example of the detail possible we are going to create a multi-colored boat using four ANTIC mode 4 characters. The boat is 8 pixels high by 16 pixels wide and consists of three colors; yellow, red and green. The black background is actually the fourth color in our character. Each group of 4 horizontal



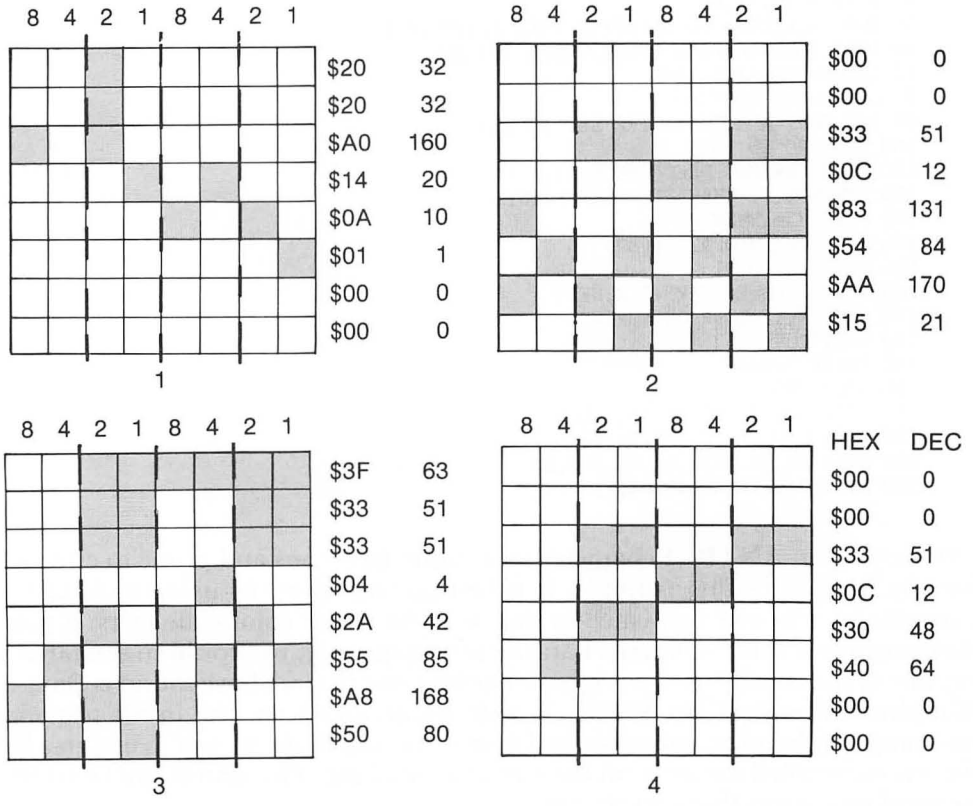
pixels makes up an individual character. Each column of colored pixels is then expanded into a double column bit pair. These eight columns make up a normal byte of character data. Each color pixel is then translated into its appropriate bit pairs so that they point to the proper color registers. Green pixels set the low bit of the pair, red pixels set the high bit of the pair, and yellow pixels set both bits. No bits are set when using the background color register. While the background is blank in our example, it can be set to a non-black color if the background register is part of the character shape. This occurs quite frequently if you are producing a map from a character set.



BLACK	712,0	PLAYFIELD 4
GREEN	708,216	0
RED	709,52	1
YELLOW	710,250	2

The first column of the first character of our ship contains only one red pixel in the second row. Since the left column of the character uses the two high bits of each byte, only the high bit is set in the second row. If the pixel would have been yellow, both high bits would have been set in that row. You will notice that after the appropriate bits in the bit pairs have been set for each character, the resultant pattern doesn't necessarily resemble the original image. Once you have finished the translation the

final character data is then obtained by adding up all of the set bits in each byte as you would ordinarily do if they were single color characters.



It is very easy to modify a BASIC 0 (ANTIC 2) display list to produce an ANTIC 4 screen. Both text modes consist of 24 rows of 40 characters. Only the interpretation of the character data by ANTIC is different. Since both display lists are identical in length, all that has to be changed is the LMS instruction and the 23 mode line instructions that follow. The LMS instruction for an ANTIC 4 screen is 64+4 =68. This follows the three instructions that blank 8 scan lines. Therefore we need only POKE this new value into location DLIST+3 where DLIST is the beginning of the display list. The following two bytes in the display list are just the address of the first byte of screen data. We ignore these two bytes. The 23 bytes following this are mode instructions. We need only change these to the value 4 in a FOR-NEXT loop in which we go from DLIST+6 to DLIST+28

The rest of the example is essentially the same as in the pumpkin example. We copy the character set into RAM, change the character set pointer, then read in the four ANTIC 4 characters. They are placed in the 97th -100th internal character positions so that they represent the letters "a" through "d". The ship is placed on the screen by printing the string "abcd" at position 5,5.

```
0 NMEMTOP=PEEK(106)-4
10 POKE 106,NMEMTOP
20 GRAPHICS 0
30 DLIST=PEEK(560)+256*PEEK(561)
40 POKE DLIST+3,68
50 FOR I=DLIST+6 TO DLIST+28:POKE I,4:NEXT I
60 POKE 708,216:POKE 709,52:POKE 710,250
70 CHROM=PEEK(756)*256
80 CHRAM=NMEMTOP*256
90 REM COPY ROM CHARACTER SET TO RAM
100 FOR I=0 TO 1023
110 POKE CHRAM+I,PEEK(CHROM+I):NEXT I
120 REM MODIFY CHARACTER SET POINTER
130 POKE 756,NMEMTOP
140 START=NMEMTOP*256+(8*97)
150 FOR I=0 TO 31
160 READ A:POKE START+I,A:NEXT I
170 REM PRINT SHIP ON SCREEN
180 POSITION 5,5
190 PRINT "abcd"
200 GOTO 200
210 DATA 32,32,160,20,10,1,0,0
220 DATA 0,0,51,12,131,84,170,21
230 DATA 63,51,51,4,42,85,168,80
240 DATA 0,0,51,12,48,64,0,0
```

Designing an ANTIC 4 character set is quite laborious and prone to error when done by hand as in this example. It is best accomplished by using a character set generator like the one furnished in this book for single color characters. If you are planning to use multi-colored character sets frequently, we would suggest that you buy one of the commercial packages. The best one that we have found is Datasoft's "Graphic Generator" for $24.95. It will design character sets in all text modes including single color, two color and four color sets. It allows you to design a block of characters called a matrix on the screen at one time. The matrix can be set to any group of characters that you choose.

Character Graphics Animation

There are two methods for achieving character set animation. The easier, but least memory-efficient, method is to rotate through a series of different character sets. The characters that are printed to the screen change, if there are distinct differences between the character data for a particular character number in each of the sets. You don't have to reprint the individual characters to the screen since the animation occurs by substituting the character data from another set rather than by changing the character itself. The change requires only a single POKE to the character set pointer at decimal location 756. The disadvantages is that each set required 1K of memory. Since animation sequences require at least 4K of memory to store the various character sets. This is a waste of space unless you are using a large number of characters. In that event this method certainly becomes the best one.

Rotating Character Sets

We have written a very simple example to illustrate the simplicity of the technique. The example rotates through four different character sets. In order to create the first three from the ROM character set, we just offset them by varying degrees during the copy stage. The goal was to create different sets in which the four graphics characters, which have a small dot in four different corners, line up. These four characters are internal characters numbers 9,11, 12, and 15. If we offset the lowest set in memory by six characters of forty-eight bytes, internal character #9 from that set will line up with character #15 from the ROM set. The other two sets need to be offset by only two and three characters respectively for the graphics symbols to all line up.

```
ROM    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
RAM1           0 1 2 3  4  5  6  7  8  9
RAM2         0 1 2 3 4 5  6  7  8  9 10 11
RAM3       0 1 2 3 4 5 6  7  8  9 10 11 12
```

Now when you print CHR$(15), internal character #15, the equivalent of internal character #9, will appear when the character set pointer points to the set at RAM1, and at #11 when using the RAM2 set, and at #12 when using the RAM3 set. This arrangement also causes internal numbers 17, 19, 20, and 23 to line up in the various sets when you print CHR$(23).

The various sets are copies into RAM memory in an area made safe by effectively lowering the actual top of memory by twelve pages or 3K. The starting address is set at the new top of memory and the others are offset by 1K each. Since the first character set is offset by eight characters or forty-eight bytes, it is best to zero out these missing characters or garbage may appear in the 0th character, which is a blank or null character, and clutter most of the screen in one or more of the RAM character sets. Copying the sets takes thirty seconds, so be patient.

The actual animation is remarkably simple once you have printed several characters to the screen. You just cycle through the various frames by changing the character set pointer at decimal location 756 ($2F4) to each of the four different character sets. A brief delay is placed between set changes, of the animation would be too fast to see.

```
0 NMEMTOP=PEEK(106)-12
10 POKE 106,NMEMTOP
20 GRAPHICS 0
30 REM CALCULATE STARTING ADDRESS FOR 4 CHARACTER SETS
40 CHROM=PEEK(756)*256
50 CHRAM1=NMEMTOP*256
60 CHRAM2=CHRAM1+1024
70 CHRAM3=CHRAM2+1024
80 REM CALCULATE HIGH BYTE OF 4 CHARACTER SETS
90 C1H=NMEMTOP
100 C2H=C1H+4
```

```
110 C3H=C1H+8
120 C4H=PEEK(756)
130 PRINT "       COPYING CHARACTER SETS"
140 REM COPY ROM CHARACTER SETS TO RAM
150 FOR I=0 TO 975
160 POKE CHRAM1+48+I,PEEK(CHROM+I):NEXT I
170 FOR I=0 TO 991
180 POKE CHRAM2+32+I,PEEK(CHROM+I):NEXT I
190 FOR I=0 TO 999
200 POKE CHRAM3+24+I,PEEK(CHROM+I):NEXT I
210 REM CLEAR MISSING CHARACTERS TO ZERO
220 FOR I=0 TO 47:POKE CHRAM1+I,0:NEXT I
230 FOR I=0 TO 31:POKE CHRAM2+I,0:NEXT I
240 FOR I=0 TO 23:POKE CHRAM3+I,0:NEXT I
250 REM PRINT CHARACTERS TO SCREEN
260 POSITION 8,8:PRINT CHR$(15)
270 POSITION 19,8:PRINT CHR$(23)
280 POSITION 30,8:PRINT "7"
290 REM ROTATE CHARACTER SETS
300 POKE 756,C1H
310 FOR DE=1 TO 200:NEXT DE
320 POKE 756,C2H
330 FOR DE=1 TO 200:NEXT DE
340 POKE 756,C3H
350 FOR DE=1 TO 200:NEXT DE
360 POKE 756,C4H
370 FOR DE=1 TO 200:NEXT DE
380 GOTO 300
```

Animation Using Different Characters

The second and more common method of achieving character set animation is to rewrite different characters to the screen in the same location. Of course this either requires a new print statement each time or a new character value POKEd directly into screen memory. The advantage of this method is that you can have as many variations as you like in the animation cycle without requiring many different character sets.

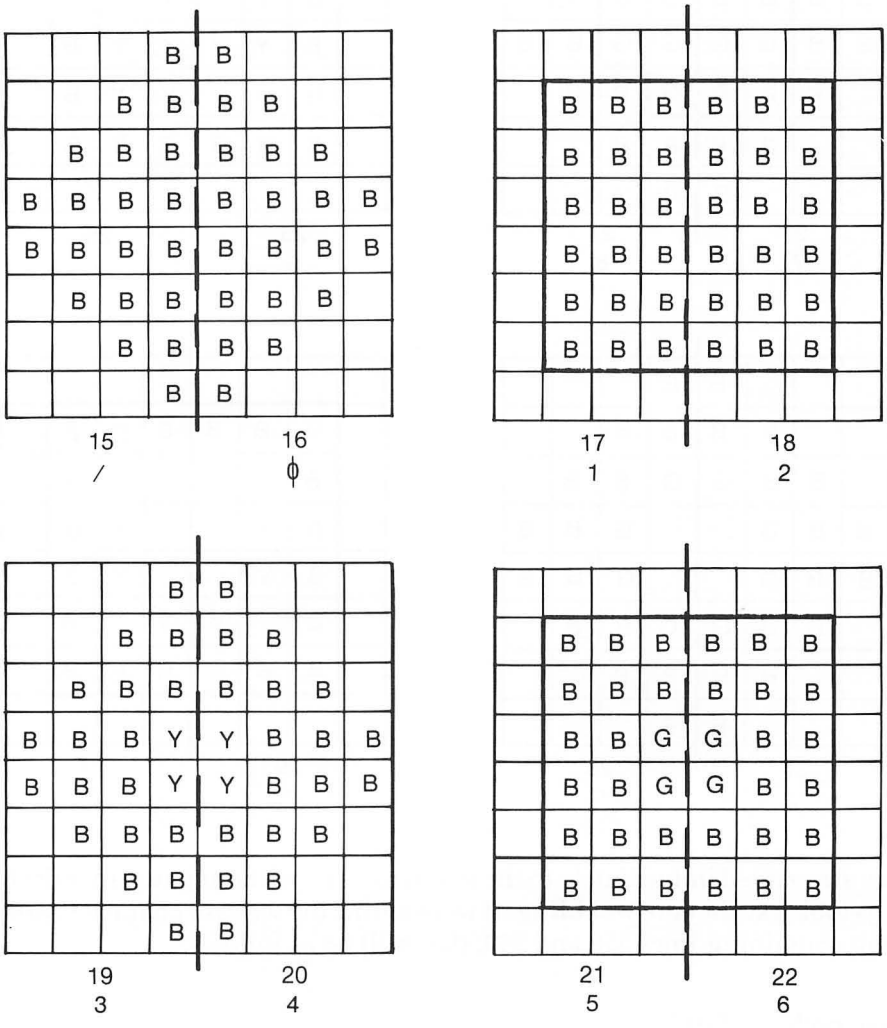
As a first example we have designed a simple ANTIC 4 character demonstration that places a random number of eggs on the screen. Each of these eggs begins to hatch into one of five different bug shapes before it eventually explodes. The complete life to death cycle requires thirteen animation frames, or fourteen, if you count the blank character needed to erase it at the end. The eggs, insects, and explosions consist of character pairs set side by side, and they are basically printed to the screen as character strings in a fixed position.

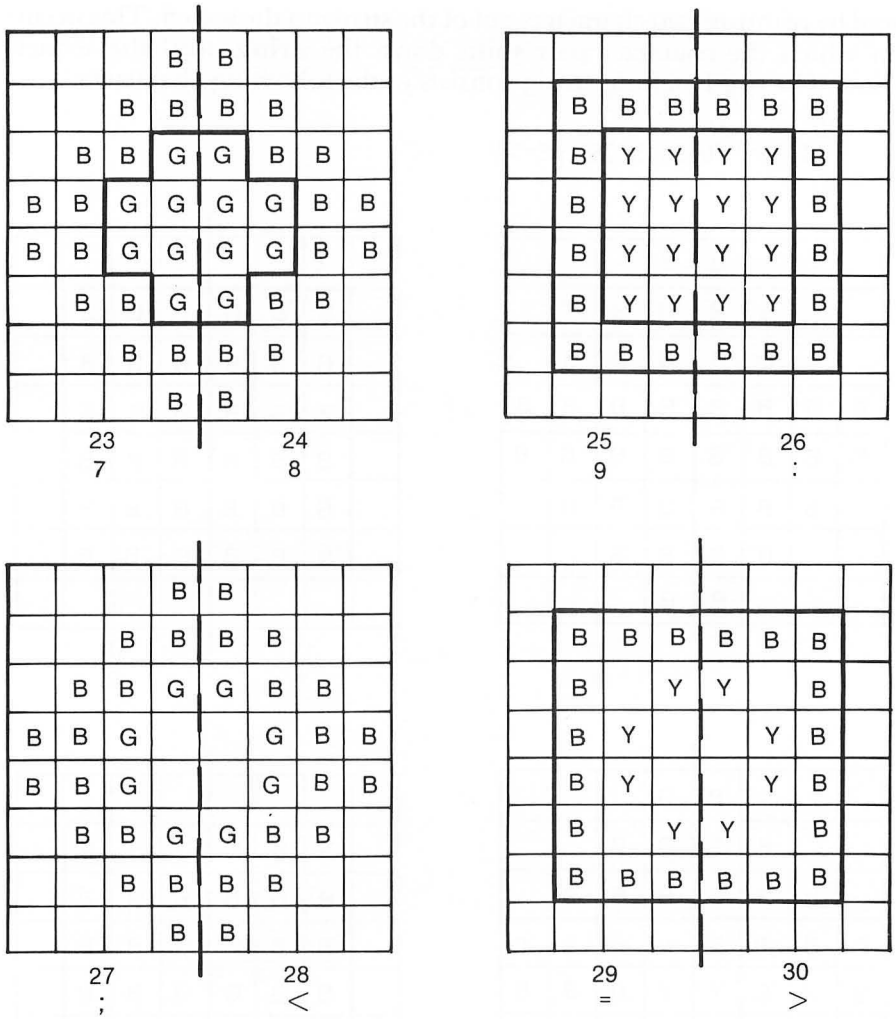
The program setup is quite similar to earlier examples. Space is reserved for the RAM character set by moving the top of memory downward, and a Graphics 0 display list is modified to ANTIC 4. Once the characters are read in through data statements and written into the RAM character set, they are transferred into strings to facilitate printing them quickly.

H$, BUG$, and BOOM$, contain the various character pairs for hatching, bug shape, and explosions respectively. Both the hatching and explosion animation is

produced by printing two characters out of the string to the screen. This occurs in a loop in which the character pair shifts down the string until the sequence is completed. The egg-hatching string consists of the following characters;

H$ = "/0123456789:;<=>"

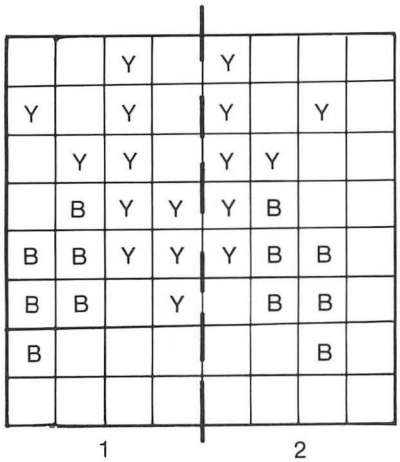


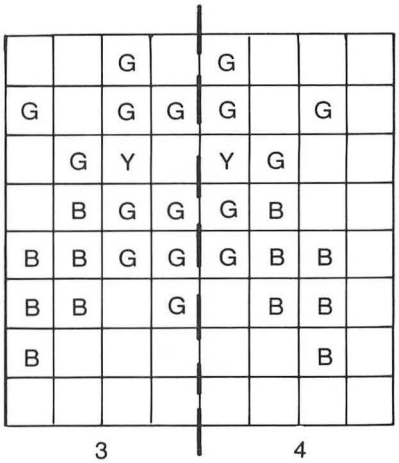


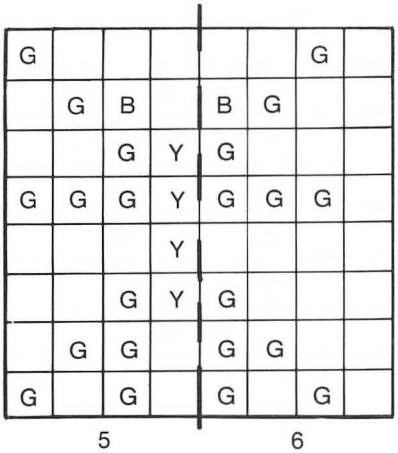
They are printed immediately to the screen as pairs without pause in order to have the eggs look like they are wobbling. If we examine the series of characters printed in the loop containing lines 330 and 340, they will be as follows:

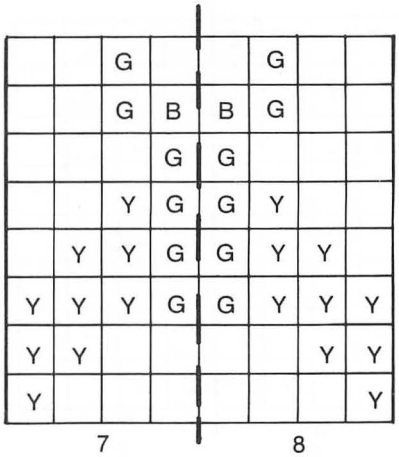
1st cycle	"/0"
2nd cycle	"12"
3rd cycle	"34"
4th cycle	"56"
8th cycle	"=>"

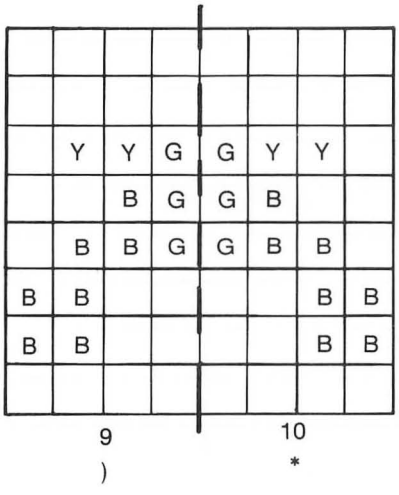
There are five random bug pairs contained in the string BUG$. BUG$ = "!"#$%&'()*". Lines 370-379 print one random bug on the screen.

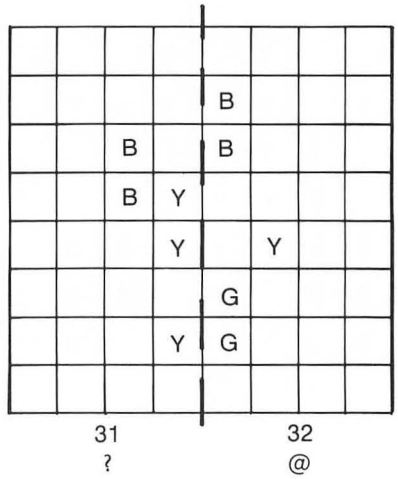

1
!
2
"

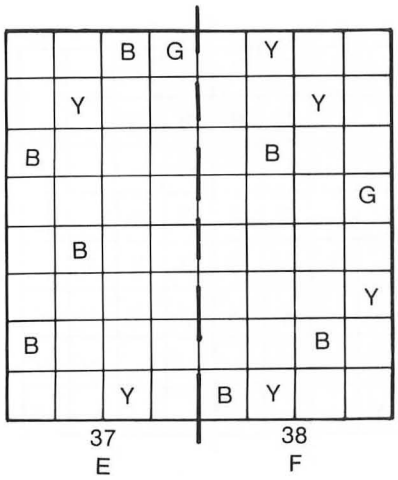

3
#
4
$

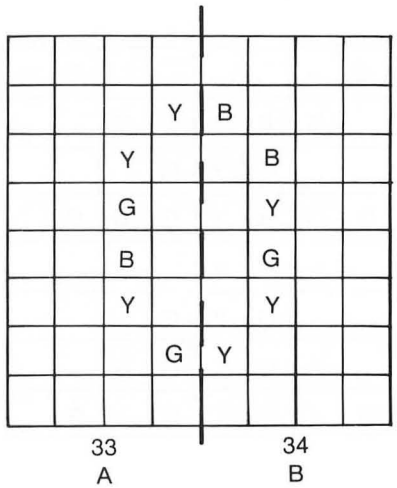

5
%
6
&

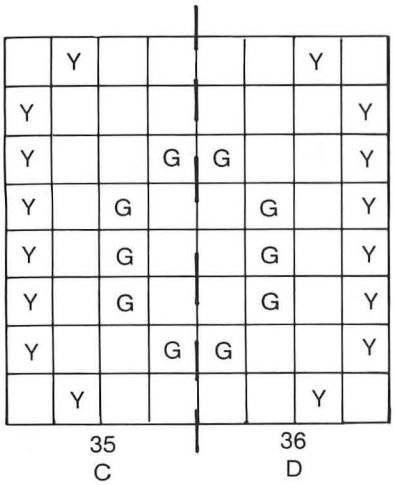

7
'
8
(

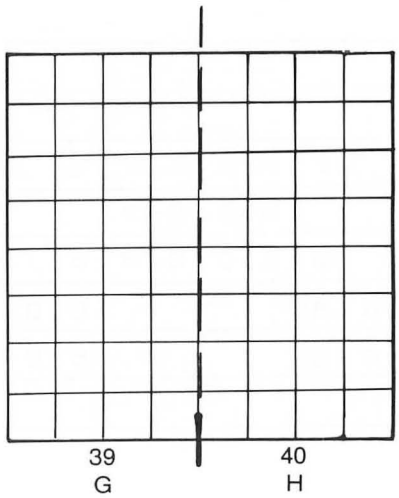

9
)
10
*


B
B
B Y
Y Y
G
Y G
31
?
32
@


B G Y
Y Y
B B
G
B
Y
B B
Y B
37
E
38
F


Y B
Y B
G Y
B G
Y Y
G Y
33
A
34
B


Y Y
Y Y
Y G G Y
Y G G Y
Y G G Y
Y G G Y
Y G G Y
Y Y
35
C
36
D


39
G
40
H

The routine that steps through the character strings BOOM$ to print each of the character pairs that make up the explosion sequence is quite similar to the egg-hatching one. There is a little more lag, since we are only printing one set at a time. If we examine the sequence of characters printed in the loop in line 460 they are as follows:

BOOM$ = "?@ABCDEFGH"

1st cycle "?@"
2nd cycle "AB"
3rd cycle "CD"
4th cycle "EF"
5th cycle "GH"

After the bugs disappear, a caterpillar-like bug dashes across the screen. This is the same shape as the ship from our previous ANTIC 4 example in the last section of this chapter. This shape is four characters wide. The string M$ = " +, -." contains a blank as the first character so that is erases the left character of the shape as it moves rightward across the screen. Essentially we print the string on top of the previous one but shifted slightly to the left. As we repeat this pattern in a continuous loop, the result is animation.

```
10 REM CHARACTER DEMO - DAN PINAL
15 POKE 106,PEEK(106)-8:SET=(PEEK(106)+4)*256:GRAPHICS 0:POKE 756,SET/256:POKE 752,
20 DLIST=PEEK(560)+256*PEEK(561):POKE DLIST+3,68
25 FOR L1=DLIST+6 TO DLIST+28:POKE L1,4:NEXT L1
30 FOR L1=0 TO 327:READ X:POKE SET+L1,X:NEXT L1
40 DIM COUNT(20,1),BUG$(10),H$(16),BOOM$(10),E$(2),M$(5)
50 FOR L1=33 TO 42:BUG$(L1-32)=CHR$(L1):NEXT L1
60 FOR L1=47 TO 62:H$(L1-46)=CHR$(L1):NEXT L1
70 BOOM$="?@ABCDEFGH":E$=H$
80 M$=" +,-."
90 POKE 708,216:POKE 709,52:POKE 710,250
200 COUNT=INT(RND(0)*20+1)
210 FOR L1=1 TO COUNT
220 H=INT(13*RND(0))*2+5:COUNT(L1,0)=H
230 V=INT(17*RND(0)+1):COUNT(L1,1)=V
240 IF L1=1 THEN 280
250 FOR L2=1 TO L1-1
260 IF COUNT(L1,0)=COUNT(L2,0) AND COUNT(L1,1)=COUNT(L2,1) THEN POP :GOTO 220
270 NEXT L2
280 POSITION H,V:? E$:NEXT L1
290 FOR L1=1 TO COUNT
300 X=COUNT(L1,0):Y=COUNT(L1,1)
310 FOR L2=1 TO 4
320 FOR L3=1 TO 4
330 POSITION X,Y:? H$((L2-1)*4+1,(L2-1)*4+2)
340 POSITION X,Y:? H$((L2-1)*4+3,(L2-1)*4+4)
350 SOUND 0,155-L2,10,8+L2:SOUND 1,155-L2,10,8+L2:POKE 53768,192:NEXT L3:NEXT L2
360 SOUND 0,0,0,0:SOUND 1,0,0,0
370 CHAR=INT(5*RND(0)):CHAR=CHAR*2+1
380 POSITION X,Y
390 ? BUG$(CHAR,CHAR+1);
400 NEXT L1
410 FOR L1=1 TO COUNT
```

```
420 H=COUNT(L1,0):V=COUNT(L1,1)
430 FOR L2=0 TO 4
440 SOUND 0,20,8,14-2*L2
450 SOUND 1,227,6,14-L2
460 POSITION H,V:? BOOM$(L2*2+1,L2*2+2);
470 FOR W=1 TO 5:NEXT W
480 NEXT L2
490 SOUND 0,0,0,0:SOUND 1,0,0,0
500 NEXT L1
510 H=0:V=INT(20*RND(0))
520 FOR L1=0 TO 35:POSITION H+L1,V:? M$
530 SOUND 0,100+L1,12,8:SOUND 1,135-L1,12,8:POKE 53768,192
540 NEXT L1
550 SOUND 0,0,0,0:SOUND 1,0,0,0:POSITION 35,V:? "     "
560 GOTO 200
10000 DATA 0,0,0,0,0,0,0,0,4,69,20,53,245,241,192,0,64,68,80,112,124,60,12,0,8
10010 DATA 138,38,58,250,242,192,0,128,136,96,176,188,60,12,0,128
10015 DATA 44,9,169,1,9,40,136,8,224
10020 DATA 128,168,0,128,160,136,8,11,2,6,22,86,80,64,32,224,128
10025 DATA 144,148,149,5,1,0,0,22
10030 DATA 14,62,242,240,0,0,0,148,176,188,143,15,0,8,8,168,21
10035 DATA 10,1,0,0,0,0,17,4
10040 DATA 129,84,170,21,63,59,59,4,21,85,170,84,0,0,17,4,16,64
10045 DATA 0,0,3,15,63,255,255
10050 DATA 63,15,3,192,240,252,255,255,252,240,192,0,63,63,63
10055 DATA 63,63,63,0,0,252,252,252,252,252
10060 DATA 252,0,3,15,63,253,253,63,15,3,192,240,252,127,127
10065 DATA 252,240,192,0,63,63,62,62,63,63
10070 DATA 0,0,252,252,188,188,252,252,0,3,15,62,250,250,62,15
10075 DATA 3,192,240,188,175,175,188,240,192
10080 DATA 0,63,53,53,53,53,63,0,0,252,92,92,92,92,252,0,3,15
10085 DATA 62,248,248,62,15,3,192
10090 DATA 240,188,47,47,188,240,192,0,63,49,52,52,49,63,0,0
10095 DATA 252,76,28,28,76,252,0,0,0
10100 DATA 12,13,1,0,1,0,0,192,192,0,16,128,128,0,0,1,4,8,12,4,2,0,0,192,48
10110 DATA 16,32,16,64,0,16,64,66,72,72,72,66,16,4,1,129,33,33
10115 DATA 33,129,4,14,32,192,0
10120 DATA 48,0,192,4,16,4,48,2,0,1,12,208,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

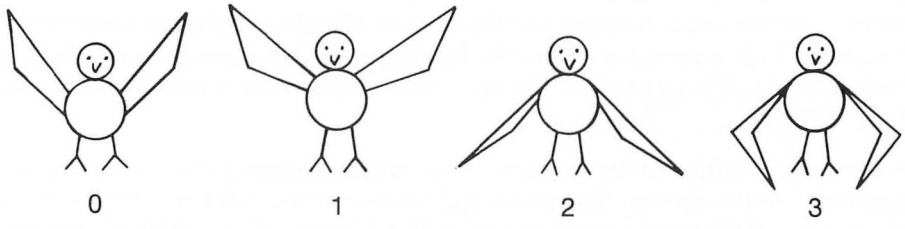
Animated Birds

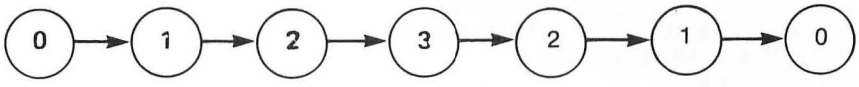
The final example is one of animated birds. The animation sequence consists of four different wing positions, so that the birds appear to be flapping their wings in flight. To give the feeling that the birds are actually hovering rather than glued in place, they moved randomly about the screen.

Each of the shapes consists of an array of characters, six across by three high. Since each shape uses it takes an array of eighteen characters, each of the four animated figures are placed eighteen characters apart in the character set.

An animation sequence like 0 -1 -2 -3 -0 -1 ...which we would call circular, would produce a discontinuity in the motion. The bird's wings are moving downward during the four cycles but if we went back to the 0th frame again, the bird's wings would suddenly be at the top. Therefore, the motion must be oscillatory so that the wings begin moving upward after reaching the botton. The sequence is 0 -1 -2 -3 -2 -1 -0 -1 -2 .....Line 40 in the program accomplishes this.

ANIMATE

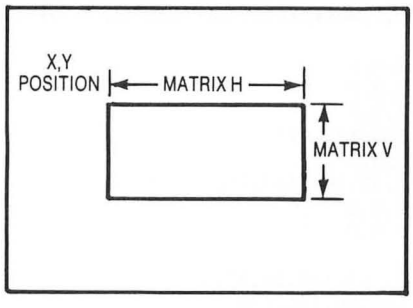




ANIMATION SEQUENCE

Printing a matrix of characters is usually tedious using normal print statements. If there is more than one row of characters involved, you will need several print and position statements. Certainly, an easier method would be to develop a Machine language subroutine in which you only had to specify the first character in the matrix, the height and depth of the matrix, and the position on the screen that you wish to print it.

The subroutine is called MCYCLER and it is stored in this example as a string, Since some users may prefer to POKE it into page six or even somewhere else, it is written as fully relocatable code. It is called with a USR statement. X = USR(MCYCLER, ROW, COL, MATRIXH, MATRIXV, STARTING CHARACTER #). MATRIXH and MATRIXV are just the horizontal and vertical size of the matrix. In the bird example they are six and three respectively. MCYCLER is the starting address of the subroutine, but since it is stored as a string, the value is ADR (MCYCLER$). The subroutine will print the matrix to the screen, except when the starting character # is zero. This feature is needed to erase the previous matrix in its last position before you draw the next frame.



Erasing one frame and drawing the next is accomplished in lines 80 and 90. The old location of the matrix is at BX (L1), BY (L1), and the new location is at NX (L1), NY (L2). Notice that the starting character during the draw stage depends on which

frame, B(L1), we are drawing, and that each matrix is eighteen characters apart in the set. The value of one is added since the 0th character is actually a blank or null character. After the subroutine draws the matrix, the old location is transferred to the new location. This is necessary since the last figure drawn must be erased from its old position BX(L1), BY(L1) before the next one is drawn at a new random position NX(L1), NY(L1).

We thought it might be instructive to see which internal characters are actually being printed to the screen. You can toggle between the ANTIC 4 bird set in RAM and the normal ROM set by pressing the SELECT key. The OPTION key resets it.

```
10 REM CHARACTER ANIMATION - DAN PINAL
15 ? "HOW MANY BIRDS (1-4)";:INPUT N:N=N-1
20 GOSUB 10000:REM INITIALIZE
30 FOR L1=0 TO N
40 B(L1)=B(L1)+F(L1):IF B(L1)=3 OR B(L1)=0 THEN F(L1)=-F(L1)
50 R=INT(3*RND(0))-1:NX(L1)=BX(L1)+R*(BX(L1)+R>0 AND BX(L1)+R<35)
60 R=INT(3*RND(0))-1:NY(L1)=BY(L1)+R*(BY(L1)+R>0 AND BY(L1)+R<20)
80 X=USR(ADR(MCYCLER$),BY(L1),BX(L1),6,3,0)
90 X=USR(ADR(MCYCLER$),NY(L1),NX(L1),6,3,B(L1)*18+1):BX(L1)=NX(L1):BY(L1)=NY(L1)
95 NEXT L1
96 IF PEEK(53279)=5 THEN POKE 756,224
97 IF PEEK(53279)=3 THEN POKE 756,CB
100 GOTO 30
10000 DATA 104,104,104,168,166,89,104,104,24,101,88,144,1,232
10005 DATA 136,48,8,24,105,40,144,248,232,208,245
10010 DATA 133,212,134,213,104,104,133,206,104,104,133,207,104,104
10015 DATA 170,160,0,145,212,201,0,240,2,232,138
10020 DATA 200,196,206,208,243,24,169,40,101,212,133,212,144
10025 DATA 2,230,213,138,198,207,208,225,96
10030 DIM MCYCLER$(72):FOR L=1 TO 72:READ X:MCYCLER$(L,L)=CHR$(X):NEXT L
10040 POKE 106,PEEK(106)-9:GRAPHICS 0:POKE 709,170:POKE 710,10:POKE 712,130
10050 POKE 752,1:CB=PEEK(106)+1:CHRSET=CB*256
10060 DLIST=PEEK(560)+256*PEEK(561)
10070 POKE DLIST+3,68:REM LMS ANTIC 4
10080 FOR L1=DLIST+6 TO DLIST+28:POKE L1,4:NEXT L1:REM CHANGE DLIST FROM GR.
      0 TO ANTIC 4
10090 POKE 756,CB:REM SET POINTER TO OUR NEW CHRSET.
10100 REM NOW POKE IN NEW CHRSET
10110 FOR L1=0 TO 583:READ X:POKE CHRSET+L1,X:NEXT L1
10200 DIM B(3),BX(3),BY(3),NX(3),NY(3),F(3)
10210 FOR L1=0 TO 3:B(L1)=L1:BX(L1)=L1*10:BY(L1)=5:F(L1)=1:NEXT L1:F(3)=-1
10220 RETURN
20000 DATA 0,0,0,0,0,0,0,0
20001 DATA 0,0,0,0,0,0,0,0
20002 DATA 2,2,2,2,2,2,2,2
20003 DATA 0,0,0,0,0,128,128,128
20004 DATA 0,0,0,0,0,0,0,0
20005 DATA 32,32,32,32,32,160,160,160
20006 DATA 0,0,0,0,0,0,0,0
20007 DATA 0,0,0,0,0,0,0,0
20008 DATA 2,0,0,0,0,0,0,0
20009 DATA 128,160,160,163,160,34,42,42
20010 DATA 0,2,2,242,194,98,106,170
20011 DATA 160,128,128,128,0,0,0,0
20012 DATA 0,0,0,0,0,0,0,0
20013 DATA 0,0,0,0,0,0,0,0
20014 DATA 0,0,0,0,0,0,0,0
```

```
20015 DATA 42,10,9,6,4,4,4,17
20016 DATA 170,168,152,164,4,4,4,17
20017 DATA 0,0,0,0,0,0,0,0
20018 DATA 0,0,0,0,0,0,0,0
20019 DATA 0,0,0,32,8,2,2,0
20020 DATA 0,0,0,0,0,0,128,160
20021 DATA 0,0,0,0,0,0,0,3
20022 DATA 0,0,0,0,0,0,0,240
20023 DATA 0,0,0,0,0,0,0,2
20024 DATA 0,0,0,2,8,32,160,128
20025 DATA 0,0,0,0,0,0,0,0
20026 DATA 40,42,10,2,0,0,0,0
20027 DATA 0,2,130,170,170,42,9,6
20028 DATA 192,96,96,170,170,170,152,164
20029 DATA 10,42,168,160,128,0,0,0
20030 DATA 0,0,0,0,0,0,0,0
20031 DATA 0,0,0,0,0,0,0,0
20032 DATA 0,0,0,0,0,0,0,0
20033 DATA 4,4,4,17,0,0,0,0
20034 DATA 4,4,4,17,0,0,0,0
20035 DATA 0,0,0,0,0,0,0,0
20036 DATA 0,0,0,0,0,0,0,0
20037 DATA 0,0,0,0,0,0,0,0
20038 DATA 0,0,0,0,0,0,0,0
20039 DATA 0,0,0,3,0,2,2,10
20040 DATA 0,0,0,240,192,96,96,168
20041 DATA 0,0,0,0,0,0,0,0
20042 DATA 0,0,0,0,0,0,0,0
20043 DATA 0,0,0,0,0,0,0,0
20044 DATA 0,0,2,2,10,8,40,160
20045 DATA 42,170,169,134,4,4,4,17
20046 DATA 170,170,154,164,4,4,4,17
20047 DATA 0,128,160,160,40,40,10,2
20048 DATA 0,0,0,0,0,0,0,0
20049 DATA 0,2,2,2,0,0,0,0
20050 DATA 128,128,0,0,0,0,0,0
20051 DATA 0,0,0,0,0,0,0,0
20052 DATA 0,0,0,0,0,0,0,0
20053 DATA 2,0,0,0,0,0,0,0
20054 DATA 128,160,32,32,0,0,0,0
20055 DATA 0,0,0,0,0,0,0,0
20056 DATA 0,0,0,0,0,0,0,0
20057 DATA 3,0,2,2,10,42,42,169
20058 DATA 240,192,96,96,168,170,170,154
20059 DATA 0,0,0,0,0,0,0,128
20060 DATA 0,0,0,0,0,0,0,0
20061 DATA 0,0,0,0,0,0,0,0
20062 DATA 0,0,2,2,2,2,2,0
20063 DATA 166,164,164,132,145,128,128,160
20064 DATA 166,6,6,4,17,0,0,2
20065 DATA 128,128,160,160,160,160,128,128
20066 DATA 0,0,0,0,0,0,0,0
20067 DATA 0,0,0,0,0,0,0,0
20068 DATA 0,0,0,0,0,0,0,0
20069 DATA 160,32,32,40,40,8,8,2
20070 DATA 2,2,2,10,10,8,8,32
20071 DATA 128,0,0,0,0,0,0,0
20072 DATA 0,0,0,0,0,0,0,0
```

CHAPTER 4

ASSEMBLY LANGUAGE APPLIED TO GAME DESIGN

The words, Machine language and/or Assembly language, evoke visions of indecipherable code to the novice BASIC programmer. The code looks unfamiliar. But so was BASIC when you were first learning it. While BASIC has its roots in the English Language and algebraic expressions, Assembly language appears to consist of unfamiliar op codes or mnemonics that are used in conjunction with an unfamiliar base 16 number system called hexadecimal.

It is our intent in this chapter to teach you the fundamentals of Assembly language programming by comparing it to similar code written in BASIC. Rather than teach you all aspects of the language, we will concentrate only on the operations needed to do simple game graphics.

A good Assembler is needed to write Assembly language programs. An assembler merely translates mnemonics like JMP, which is equivalent to a GOTO, into hexadecimal opcodes that the computer understands. Most Assemblers have an editor, an Assembler, and a debugger. The editor allows you to enter Assembly language code usually by line number and later edit, delete, or insert particular lines. The Assembler portion converts your source listing into Machine Code in a two-pass operation. Since any line of code can have a label in its first field, the Assembler will automatically calculate the branches or GOTOs to lines referenced with these labels. Also, if you want to store a variable called ZAP, the Assembler which assigns a memory storage location for the variable will automatically furnish the correct memory address for any subsequent store or load operations using that variable. Last, there is a Machine language monitor or debugger that helps locate errors. It allows you to examine and change both memory and internal registers. It also includes step and trace features that allow you to step through your code one instruction at a time.

Readers who already own assemblers may use the one they have. We have provided a translation table in the Appendix in the back of this book to aid you in converting our SYNASSEMBLER source code to that used in your assembler. We chose SYNASSEMBLER when we began this book in the Spring of 1983 because it was co-resident (screen editor, assembler, and debugger are in memory simultaneously) and was available in cartridge form.

For those of you who are new programmers, or are unhappy with their present assembler, we recommend either the *F-S Macro Assembler 40/80* from Stanton Products (See coupon in back of book), an enhanced disk version of the now discontinued *SYNASSEMBLER*, or *MAC 65* from Optimized Systems Software. Both of these assemblers are fast (2000 lines/minute), are co-resident assemblers,

allow source files to be chained, and offer a choice of assembling to either disk or memory. Both of these are professional packages and are used as development tools in various software houses. The *F-S Macro Assembler 40/80* and the discontinued *SYNASSEMBLER* are both derived from the S-C family of assemblers on the Apple II computer. Whereas the new *F-S Macro Aseembler 40/80* is compatible with the new XL series of computers, unpatched versions of *SYNASSEMBLER* are not. The *F-S Macro Assembler 40/80* is completely compatible with *SYNASSEMBLER* source files with the exception of the way it handles ATASCII string data. A simple global replace will suffice. (see note in Appendix on assemblers differences.)

Our readers will certainly want to know why we don't use the more popular *Atari Editor Assembler* cartridge. First, it is very, very slow, often taking ten minutes to assemble a 1000 line program. Second, it doesn't allow chaining of files, nor assembly to the disk. Third, it is full of bugs. It remains popular mostly to beginner programmers who want to try to write a very short Assembly language subroutine that will interface to their BASIC programs.

Basic Assembly Language

The Atari computers contain a central processing unit (CPU), a 6502A microprocessor that operates at 1. 8 Mhz. It accepts instructions to perform various operations, like taking a value and storing it somewhere in memory, adding a number to another number located in one of its internal registers, or comparing two values. What makes programming in Assembly language rather difficult (or at least tedious) is that the computer can only execute one tiny instruction at a time, and only perform its operations in three internal registers. These three addressable registers are known as the X register, Y register, and Accumulator. Each can hold eight binary digits called bits, which are individually valued at 0 or 1. The eight bits, collectively called a byte, have values ranging from 0 to 255 decimal or ($00 to $FF in hexadecimal notation).

Essentially, the computer, which is an eight-bit microprocessor, can manipulate data whose values range from all eight bits off (00000000) to all eight bits on (11111111). The average person has great difficulty in thinking of values represented by 0's and 1's. Fortunately, someone invented a number system called hexadecimal, which is base 16 instead of binary or base 2.

Hexadecimal Numbers

Since 16 is 2x2x2x2, we can divide our eight bits into two four-bit groups. If you determine each of the decimal equivalents of all the combinations of base two representations, you obtain the following table. These values range from 0 to 15 decimal. In the hexadecimal numbering system, values above 9 are represented by the letters A-F. In order to prevent confusion between decimal and hexadecimal numbers, hexadecimal numbers are preceded by a "$".

BINARY	DECIMAL	HEXADECIMAL
0000	0	$0
0001	1	$1
0010	2	$2
0011	3	$3
0100	4	$4
0101	5	$5
0110	6	$6
0111	7	$7
1000	8	$8
1001	9	$9
1010	10	$A
1011	11	$B
1100	12	$C
1101	13	$D
1110	14	$E
1111	15	$F

Hexadecimal numbers are very much like decimal numbers. They can be added and subtracted in like manner. The only difference is that instead of having units, tens, hundreds, etc, the hexadecimal numbers have units, sixteens, 256's, and so forth. Each successive digit is sixteen times the position to the right instead of ten times as in our decimal system.

DECIMAL	HEXADECIMAL
1 6 5	$ 1 3 A
1 HUNDRED	1— 256
6 TENS	3 SIXTEENS
6 ONES	A-ONES
1 x (100) = 100	1 x (256) = 256
+ 6 x (10) = 60	+ 3 x (16) = 48
+ 5 x (1) = 5	+ A x (1) = 10
165 DECIMAL	$ 13A = 312 DECIMAL

Hexadecimal numbers are used to address the Atari's 48000+ memory locations. Each group of 256 bytes ($00 - $FF) is called a page, starting with page zero. In 48K Atari computers, memory is directly addressable from locations $0000 to $BFFF (0 -49151). Locations above $BFFF are also addressable, but these locations don't contain RAM. The area from $D000 to $D7FF contain custom hardware chips such

as the GTIA, POKEY, PIA, and ANTIC microprocessor. Some of these hardware locations can be read and some written to. The area above that, $D800 to $FFFF, contain the 10K operating system ROMS.

Memory Considerations in Assembly Language

The bottom of RAM, pages 0 thru 5 ($0000 - $05FF) are generally off limits for program storage. Zero page ($00 - $FF) is a very special area. There are a number of zero page addressing instructions that execute faster because they require only two instructions instead of the usual three. This is because they only need to address a memory location from $00 to $FF instead of $0000 to $FFFF. These locations are used extensively by the Operating System.

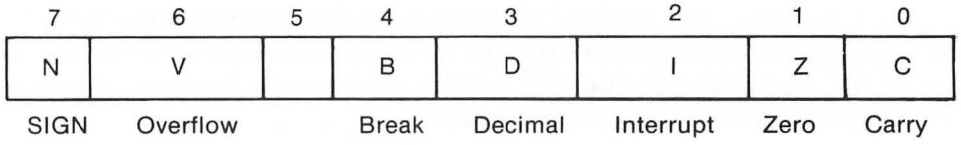
Only the last few bytes of zero page are available to the user. In fact, if you are using *Synassembler* only locations $F0 - $FF are totally free. You can also use $D6 - $EF, if you don't mind if your data is altered by the floating point package each time arithmetic operations are performed by BASIC. And if you are writing a subroutine to be accessed from BASIC, only locations $CB - $D1 (203-209) are available. $D4 and $D5 can be used to send variables back to BASIC via the USR function.

Page one of memory ($100 - $1FF) is reserved for the stack. It is used by a special purpose register in the 6502A microprocessor for keeping track of return addresses when calling subroutines. This scratch area for the Stack Pointer is sometimes used for temporary register storage.

Pages two and three are used for various I/O operations, and operating system shadow registers, page four for the cassette buffer, page five for the keyboard buffer, and pages seven through twenty-eight for DOS 2.0's file management system. Essentially the area below 7420 ($1CFC), with the exception of page six, is off limits to programmers using DOS. However, if DOS isn't resident, you can begin storing at 1792 ($700) safely. A pointer to the low end of memory, MEMLO, can be read at locations 743,744 ($2F7,2F8).

Program Counter & Program Status Word

When a microprocessor processes a Machine language program, it keeps track of which instruction it is executing with an internal 16-bit register called the program counter. The program counter contains the current address of the instruction that is being processed. When the computer finishes with an instruction, it sets a flag or condition in a 7-bit, Program Status Word, which is another register. For example, if you want to test if a value in the Accumulator is equal to zero, you compare the value in the Accumulator to zero. If this value is equal to zero, the zero flag will be set and the next instruction, Branch Equal to Zero (BEQ), will be executed. Other flags that can be set are the carry flag, and the negative flag. A diagram of the Program Status Word is shown below.



PROGRAM STATUS WORD

OP Codes

The 6502A microprocessor accepts only Machine language instructions. These are called opcodes. When the computer encounters a $4C, it performs an equivalent to a GOTO in BASIC. The Machine language instruction $4C 00 08 tells the computer to jump to memory location $800. (Remember, addresses require two bytes. The low order byte in this case contains $00 and the high order byte, $08—in effect, the reverse order of the actual values.) Unfortunately, Machine language is difficult to remember, so programmers invented a substitute called Assembly language, wherein each opcode is assigned a mnemonic such as JMP, BRK, or LDA. The above example looks like this: JMP $0800.

If you were to type the following Machine Code into the monitor in your Assembler, you would see how the monitor disassambler interprets the code, as in the following example:

4000: A9 30 8D 00 41 CE 00 41 AD 00 41 C9 00
D0 F6 60 <CR>

If you enter a 4000L from the Synassembler monitor you will see the following:

```
4000: A9 30     00030          LDA #$30
4002: 8D 00 41  00040          STA $4100
4005: CE 00 41  00050          DEC $4100
4008: AD 00 41  00060          LDA $4100
400B: C9 00     00070          CMP #$00
400D: D0 F6     00080          BNE $4005
400F: 60        00090          RTS
```

The disassembler translates the Machine Code to more easily understood mnemonics. In the first line of code, LDA is the mnemonic for Load Accumulator. It is the instruction for the 6502 to load the Accumulator with an immediate value—in this case, $30. The # sign signifies that it is an "immediate" instruction; the ($30) is the data portion of the instruction. The STA in line two is an "absolute" instruction. It specifies the address in memory for storing the byte of data that is in the Accumulator.

The difference between "immediate" and "absolute" instructions is an important point. Let us take the example LDA #$30. In this "immediate" instruction, the computer takes the operand ($30) as a value and places it in the Accumulator.

However, LDA $30 is an "absolute" instruction, so the computer takes the operand as an address from which to load data into the Accumulator. In both cases, we get a value in the Accumulator. You can tell the modes apart because "immediate" instructions have a # sign before the operand.

You might wonder, what does this code do? It is a time delay subroutine. It puts a decimal 48 in memory location $4100. Line two stores it there, then the value stored at that memory location is decremented by one in line three. It is then reloaded into the Accumulator to be compared against the value zero. If it is zero it falls through to the return-from-subroutine instruction and ends; but if it isn't zero it branches back to memory location $4005. That location tells the computer to decrement the value in $4100 once again. The code will perform this small loop until the value in $4100 becomes zero. At that time, the test for a zero becomes true and the program returns to the line after the JSR in the program that called it.

Does it work? First type 400E:00 <CR> to change the RTS to a BRK. This will return us to the monitor when we are finished. Then type 4100:AA <CR> to place something in that memory location so that if you look at it later you will believe the program did something. Finally, do type 4000G <CR> to start the routine. The code returns you back to the monitor when it finishes a split second later. Now type 4100 <CR> and a 00 is returned. This is the value in memory location $4100. You can do a 4000S <CR> and an S <CR> each time to watch the code single step, or you can trace the entire operation by typing a 4000T <CR>. The strange numbers that appear below each line of code are the values in the internal registers. A is for Accumulator, X for X register, Y for Y register, P is the Program Status Word, and S is the Stack Pointer.

This program has a direct analogy to the following BASIC program:

```
10 X=48
20 X=X-1
30 IF X<>0 THEN 20
40 RETURN
```

The major differences between the two programs is that in Assembly language there are no line numbers used within the code (line numbers are used only by the editor to place your text in order, and you have to take care of every minute detail. BASIC automatically assigns the storage locations of all variables and the location of each instruction in memory. In Assembly language programming, we have to assign the X variable to memory location $4100, and have to calculate the relative branch or GOTO so that it references the memory location $4005. This is done by branching back $F6 bytes or -8 bytes to the proper address. Yet many of these details can be greatly simplified if we use an Assembler to do our programming.

The same program using an Assembler looks like the following:

```
LINE LABEL INSTRUCTION COMMENT
#    FIELD FIELD       FIELD

00010          .OR $4000    ;ASSEMBLE CODE AT $4000
00020 X        .EQ $4100    ;X IS STORED AT $4100
00030          LDA #$30
00040          STA X
00050 LOOP     DEC X        ;X=X-1
00060          LDA X
00070          CMP #$00     ;DONE?
00080          BNE LOOP
00090          RTS
```

The Assembler generates identical Machine Code, but many of the tedious details are simplified. Once X is equated to the memory location in line 2, references to that variable in lines 4 through 6 are handled automatically. If X were assigned to a different memory location because we lengthened our program, you would only have to change line 2. Also, labels act like line numbers in BASIC. Since the Assembler assigns the line of code labeled LOOP to a particular memory location, it can calculate the correct branch automatically when it encounters line 8 during assembly. The .OR in line 1 is a pseudo-op, understood only by the Assembler. This does not generate code but tells the Assembler where the code is to be run and stored. The pseudo-op .TF causes the generated code to be stored to the disk rather than to memory.

Addressing Modes

Now that you have had a taste of Assembly language programming and have seen that it isn't as bad as you thought, there are a number of fundamental operations that must be learned. The most important operation is to move numbers from one memory location to another. This can be accomplished by loading a value into any one of three internal 6502 registers—the Accumulator, X, or Y registers—and storing that number somewhere in memory. A LDA (Load Accumulator) instruction can be carried out in several different ways depending on its addressing mode. First we can load the Accumulator with a real hexadecimal value (LDA #$05). This is called Immediate Mode Addressing. Sometimes we need to be able to load the Accumulator with a variable stored in a memory location (LDA $4100). This is called Absolute Addressing.

The only other addressing method that we will discuss for the time being is the Indexed Addressing mode. It takes the form of LDA $4100,X or LDA $4100,Y depending on whether the X or Y register is used as an index. If, for example, the X register contains a #$05, then the instruction above loads the value from location $4100 + $05 or $4105. This addressing mode is used primarily for indexing into tables stored at particular memory locations. There is no problem with the tables crossing page bounderies. For example, if your table began at $4080 and the X-register contained a $90, then the instruction LDA $4080,X would fetch the value in memory location $4080 + $90 or $4110.

EFFECTIVE ADDRESS = ABSOLUTE ADDRESS + X

EFFECTIVE ADDRESS = ABSOLUTE ADDRESS + Y

Store operations are similar to load operations. You can store a value into an "absolute" memory location, or you can store indirectly into a memory location, offset by the value contained in either the X or Y register.

In summary, the table below shows the various load and store operations.

	ACCUMULATOR	X REGISTER	Y REGISTER
LOAD	LDA #$05	LDX #$05	LDY #$05
	LDA $4100	LDX $4100	LDY $4100
	LDA $4100,X		LDY $4100,X
	LDA $4100,Y	LDX $4100,Y	
STORE	STA $4100	STX $4100	STY $4100
	STA $4100,X		STY $4100,X *
	STA $4100,Y	STX $4100,Y *	

*Both indirect operations involve zero page addressing only.

Incrementing & Decrementing

Sometimes it is necessary when counting cycles or looping through code to increment or decrement a value directly similar to a FOR-NEXT loop in BASIC. In Assembly language, either the X and Y registers or any memory location can be incremented or decremented. If the X register contained a $FE, then it would contain $FF when incremented. But if it contained a $FF, it would wrap around to become $00. The computer informs you by setting a zero flag in its Program Status Register.

	ACCUMULATOR	X-REG	Y-REG	MEMORY LOCATION
INC BY 1	NOT AVAILABLE	INX	INY	INC $4100
DEC BY 1	NOT AVAILABLE	DEX	DEY	DEC $4100

Stack Instructions

There is a special area in the computer ($100 - $1FF) that is used quite frequently by an internal register called the Stack Pointer. The computer uses this area to save return addresses when handling either interrupts or subroutines. The stack is like a dish dispenser. Bytes are pushed on the stack in order, and pulled off in reverse order. The first byte stored is the last byte to be pulled off. The Stack Pointer always points to the next free byte in the stack. Since the stack is only 256 bytes long, only 128 address pairs can be stored at any one time.

Normally the stack would be of little interest to programmers except that it can also be used to temporarily store data. If you were worried about your three registers being altered in a subroutine, you could push all three values onto the stack before calling the subroutine, and then pull them back off when you return from the subroutine. BASIC also uses the stack to transfer data in the USR function when calling a Machine language subroutine. The top byte in the stack contains the number of variables being passed. The values follow in two byte pairs in hi byte—low byte order.

Two basic Machine language instructions provide key tools for using the stack. PHA pushes the value in the Accumulator on the Stack. PLA pulls the top value of the stack and places it in the Accumulator. Since these instructions only involve the Accumulator, you would need to transfer the value in the X register to the Accumulator (TXA) in order to save the X register on the stack. Similarly you would transfer the Y register to the Accumulator (TYA) first before a PHA to the stack. Be careful when working with the stack. For instance, if you push data onto the stack while in a subroutine and don't pull it back off, when the subroutine reaches the RTS instruction it will return to the main program at the wrong address.

Altering Program Flow

Program flow can be altered, as in BASIC, with instructions that resemble GOTO, GO SUB, and IF...THEN statements. The JMP instruction is equivalent to a GOTO statement; it can transfer control to any location in the machine to continue executing code. JMP $8D6C instructs the computer to continue executing code beginning at address $8D6C. The GOSUB statement is identical to a JSR (Jump Subroutine) in Machine language. When the computer reaches the instruction $5A83, it pushes the two-byte memory address of the instruction onto the stack, so that when it returns from the subroutine via an RTS (ReTurn from Subroutine), it will know the address where it will continue the program. When it returns, it pulls the return address off the stack and increments it by one so that it points to the next executable instruction.

The IF...THEN statement is analogous to a number of branch instructions which test the Program Status Register to see which flags are set. Usually, you use compare operations to set flags. You can compare a value against the value stored in either the Accumulator, the X or the Y Registers. The mnemonics are CMP, CPX and CPY, respectively. For example,

```
LDA $4100    ;LOAD ACCUMULATOR WITH VALUE AT $4100
CMP #$05
```

Different flags are set depending on the result.

Branch instructions are very similar to a JMP instruction (which is an unconditional branch), except that only under certain circumstances will they cause program flow to continue at a different location. For example, if we were to test for that

wraparound case when we incremented the X-register that contained $FF, we would want to test the Zero Flag with a Branch Equal Zero (BEQ) instruction, and go to some label if the condition is true.

```
        LDX $4100    ;LOAD X REGISTER WITH VALUE IN MEMORY
        INX          ;INCREMENT X - REGISTER
        BEQ SKIP     ;TEST IF 0, AND IF TRUE GOTO SKIP
        RTS          ;RETURN TO MAIN PROGRAM
SKIP    LDA #$04
         .     .
         .     .
```

This short example loads a value from the memory location into the X register, then increments it. If wraparound occurs, the test for a zero flag causes the program to jump to a label called SKIP, and the code does not return to the program that called it via the RTS. There are numerous tests on each of the flags in the Program Status Register. A summary is shown below.

BCS	— Branch if the carry flag is set.	C = 1
BCC	— Branch if the carry flag is clear.	C = 0
BEQ	— Branch if the zero flag is set.	Z = 1
BNE	— Branch if the zero flag is clear.	Z = 0
BMI	— Branch if minus.	N = 1
BPL	— Branch if plus.	N = 0
BVS	— Branch if overflow is set.	V = 1
BVC	— Branch if overflow is clear.	V = 0

Most Assemblers offer alternative mnemonics for BCC and BCS. Since, during comparisons, the carry flag is set when the value in the appropriate register is equal or greater than the value compared, BCS might be called BGE (Branch Greater or Equal). Likewise, BCC is equivalent to BLT (Branch Less Than). Why use these alternatives? Because they are easier to remember and visualize, and they make it clear that you are doing logical comparisons, rather than testing the results of an addition or subtraction.

There is one other important concept that should be understood when doing comparisons. I implied that the subsequent branch was like a GOTO in BASIC or like a JMP in Assembly language. This is not entirely true, since the range of the branch cannot exceed -126 to +129 bytes. This is because the branch instruction is only two bytes long. The first byte is the instruction code and the second the relative address. It takes a two byte address to branch to any place in memory (Except Page Zero). The JMP instruction has the advantage that it is three bytes long. In most cases, this limitation will not cause problems. But if a "branch out of range error" occurs, you must reverse the test so that it will reach the required destination via a JMP instruction.

Example: If BEQ SKIP is out of range then substitute the following:

```
BNE *+$5        or   BNE B
JMP SKIP             JMP SKIP
.                    B NOP
.                    .
```

This change causes the program to drop through the JMP instruction if the zero flag was set, and then jump to location SKIP. However, if the zero flag is not set, it will advance ahead five bytes to the instruction following the JMP. All other branch instructions work in a similar manner. This gives the equivalent of a Long Branch.

Addition & Subtraction

Simple addition and subtraction of unsigned numbers is easily accomplished in Machine language. All additions and subtractions must be performed one byte at a time. Thus, large numbers or multi-byte numbers (those that exceed $FF), must be added or subtracted one byte at a time, and the carry flag must be accounted for. It's actually not much different from addition of two multi-digit decimal numbers. Those numbers have a digit in the ones column, another in the tens, etc. If you add 65 to 78, you add the ones column first. Five plus eight equals 13. The value in the ones column is 3; you then carry the one "ten" into the tens digit column before you add the two numbers in the tens column. Hexadecimal addition is similar. You clear the carry before you add. If the sum of the two values exceeds $FF, the carry is set. Since you don't clear the carry when adding the next higher byte, the resultant answer will be the sum plus the previously computed carry, as in the following example:

```
EXAMPLE:        +CARRY
                    63     F4
                  + 02   + 16
                  ---    ---
                    66     0A   ;SETS CARRY
```

The code for addition and subtractions is as follow:

ADDITIONS

```
CLC          ;CLEAR CARRY
LDA #$F4     ;LOAD LOW ORDER BYTE
ADC #$16     ;ADD WITH CARRY
STA LOW      ;STORE LOW BYTE
LDA #$63     ;LOAD HIGH ORDER BYTE
ADC #$02     ;ADD WITH CARRY (NOTE DON'T CLEAR CARRY)
STA HIGH     ;STORE HIGH BYTE
```

SUBTRACTIONS

```
SEC          ;SET CARRY FLAG
LDA #$F4     ;LOAD VALUE
SBC #$16     ;SUBTRACT WITH CARRY
STA VALUE    ;STORE RESULT
```

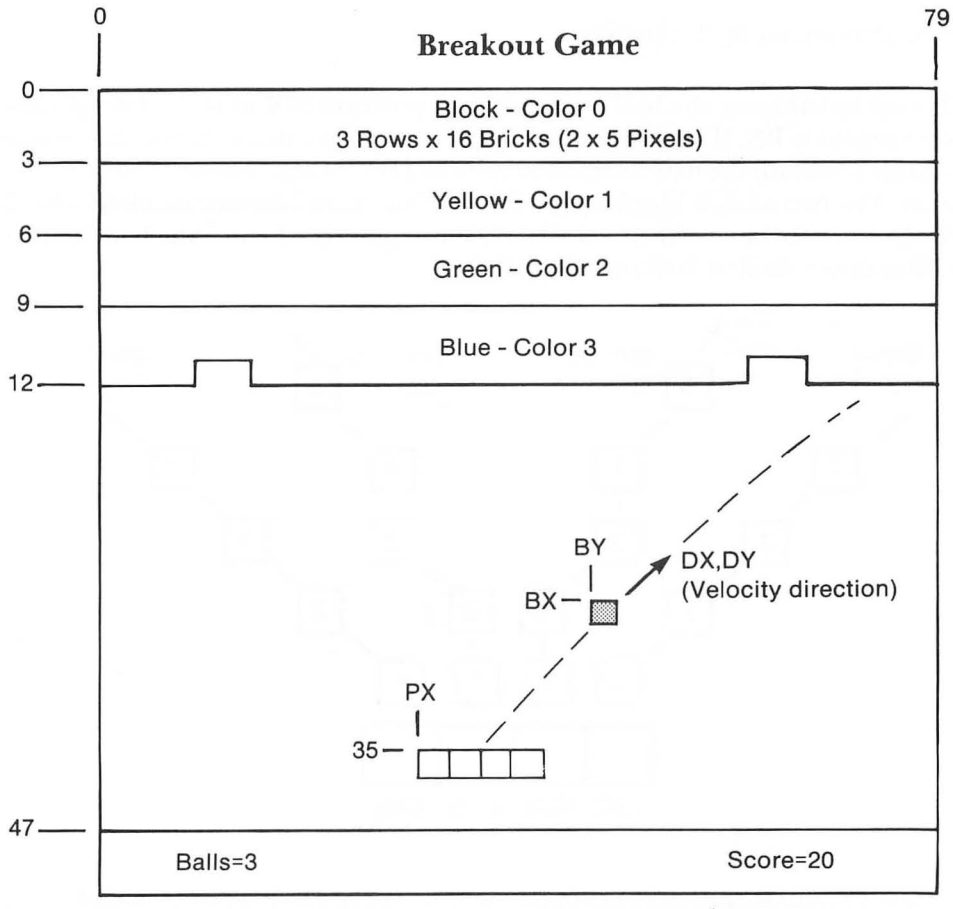
You should be aware that the rules for subtraction are different from the ones for addition. The carry must be set first. This is equivalent to a borrow in subtraction. After the subtraction operation, the carry will be clear if an underflow (borrow) occurred. The carry will be set otherwise. Setting the carry is very important, a step that many beginners forget. The results are invariably incorrect if this step is skipped—and possibly even "random," since the status of the carry flag can be on or off when the subtraction operation is performed. This can make debugging difficult.

Breakout Game (BASIC)

The "Breakout" game involves the simplest animation technique available on the Atari, moving individual pixels from one position to a new position. We have a Graphics 5 pixel-sized ball that bounces around the screen. It will ricochet off a movable paddle, the walls, or any of the 2 pixel-high by 5 pixel-wide colored bricks. Movement is accomplished by erasing the ball at its old position and redrawing it at its new position. The ball is very predictable. It changes direction only upon collision, and in all cases (except contact with the paddle) simply reverses direction. The point of contact with the joystick-controlled paddle determines the ball's direction. Balls striking the left end travel upwards and to the left at a 45 degree angle, while balls striking the inside left travel in the same direction but at a 60 degree angle. Balls striking the paddle's right side travel at similar angles, but to the right.

Once you have the design description, in this case a game that is an old classic, the next step is to translate it into a logical sequence of events and their consequences. This can best be accomplished by drawing a flow chart that shows the possible pathways for each module in the program. Each of these modules can be as small as a single statement, or can consist of entire subroutines. No matter how detailed or general you make it, the flowchart must accurately represent the game's logic. While it is a good tool for learning to think logically, a flowchart isn't necessary or required in all cases. Many good programmers have never drawn one. They obviously have the ability to flowchart unconsciously in their minds.

The game should be programmed in small steps rather than as a complete entity. This way you get to see results early. Besides, it is easier to debug a small section, such as the ball bouncing off the paddle and moving around the screen, than to attempt to debug a complete program that is full of errors. The most successful programmer will be one who can debug by watching what goes wrong on the screen.

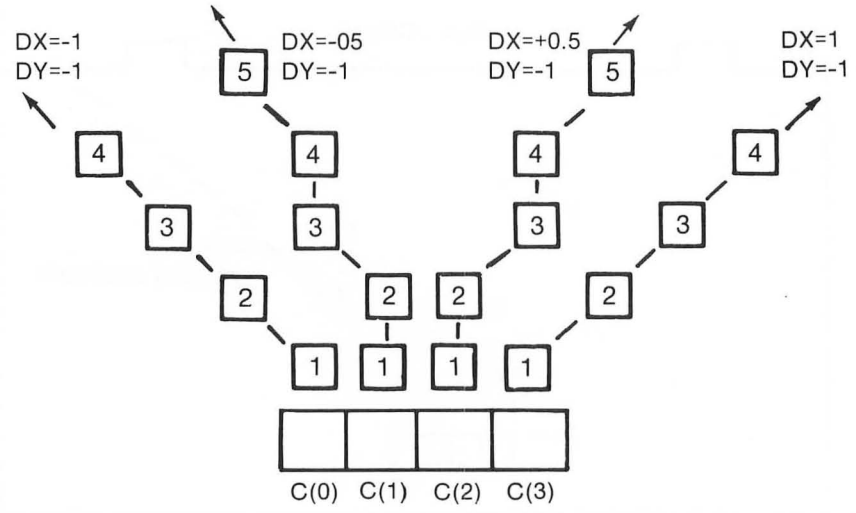


Paddle Position

Determining where the ball strikes the paddle is easy in our "Breakout" game. The paddle is always drawn two-pixels wide at row 36 decimal or $24, and the first pixel begins at PX, a variable controlled indirectly by the joystick. Actually the new paddle position is P = P + D where D depends on the direction of movement and whether the button is being pressed. If the joystick is pushed to the left, D=-1, while if it is pushed to the right, D=1. When the button is pushed, D = D*3, and the paddle moves at triple speed. The Boolean logic in line 230, ((P+D)>0 AND P+D<76) gives a value of true=1 or false=0 depending on whether the paddle has exceeded the screen bounds after movement. If it hasn't, the result is P = P+D*(1), and there is a new paddle position. If it has, the result is P = P+D(0) and the paddle remains stationary.

Ball's Position & Velocity

It is easy to compare the ball's new vertical position NX to that of the paddle's leftmost position PX. The difference NX-CX is C. You can use this value to index into a table to obtain the new horizontal velocity; DX = C(C). These values vary with position. The two outside blocks give a DX of +1 or -1, and the two inside blocks give a DX of +½ or -½. The vertical velocity, DY is equal to -1 since the ball is always travelling upwards after striking the paddle.



In order to update the ball's position, we take the old ball's position and add the change in position or its directional velocity. The format is:

NEW POSITION = OLD POSITION + CHANGE IN POSITION

NX = BX + DX
NY = BY + DY

Incrementing or decrementing the ball's position by ½ in the X direction is not physically possible since screen positions are whole numbers. The ball's position is truncated to the nearest integer value with the INT function. The result is that the ball remains stationary in the X direction during one frame, then moves one whole pixel position during the next frame or cycle.

Collisions with Bricks

As the ball bounces around the screen it will soon collide with one of the colored 2 by 5 pixel-sized bricks at the top of the screen. It is possible to test for a collision by using the LOCATE function. This function, which returns the color register at the ball's position, works only in BASIC Graphics modes 3-8. Non-zero values in this

example indicate a collision with one of the three colored bricks (Playfields #1-3).

If there is a collision, the correct block needs to be removed. This is quite simple to calculate for the X direction:

$$C = INT(NX/5)*5$$

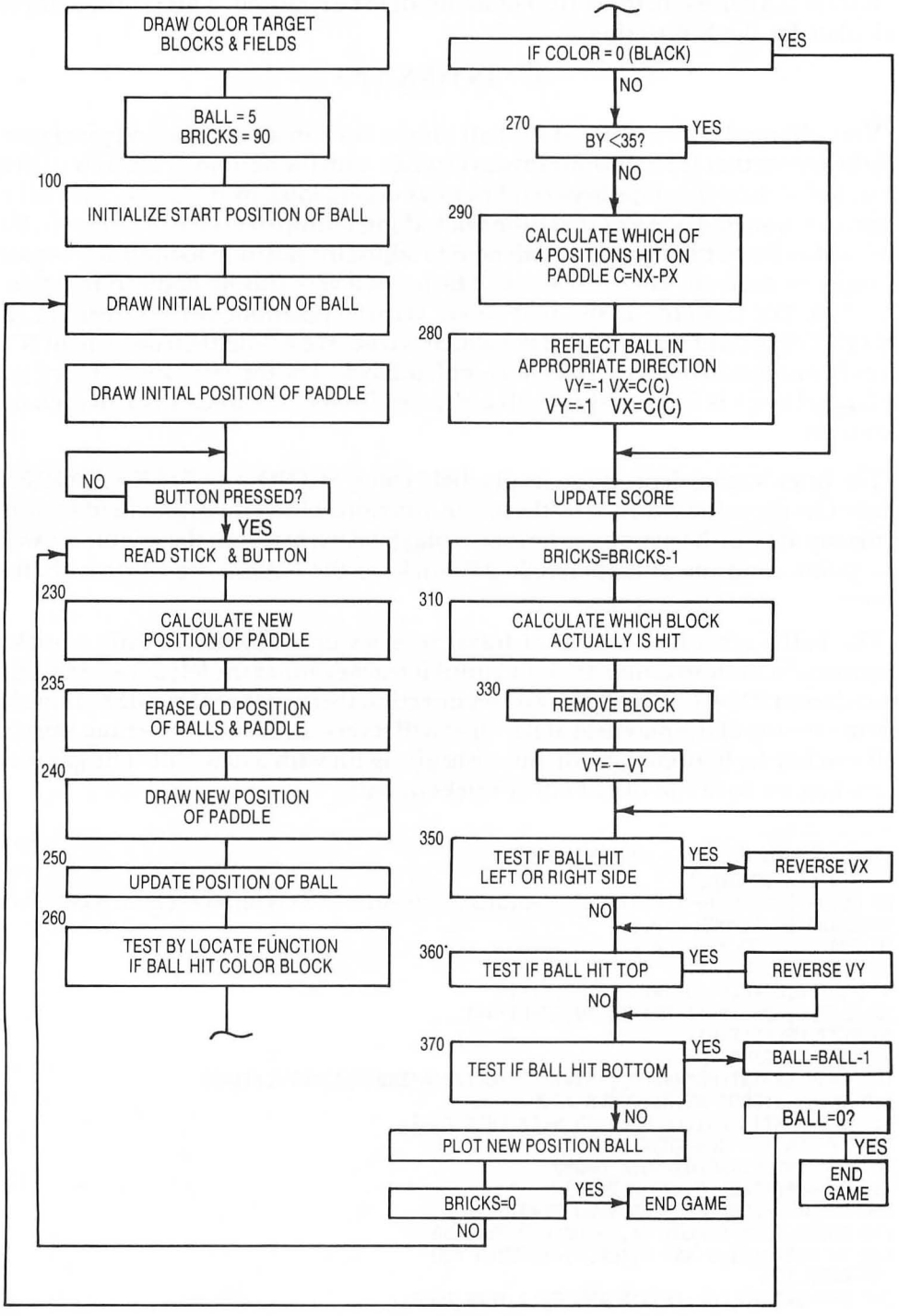
You still need to determine if the ball hit the brick in an even or odd pixel row. It might appear that the ball would always collide with the bottom or odd row of pixels first, but if there are gaps between bricks as occurs later in the game, the ball can approach from the side and strike the brick along the top or even row of pixels. If the ball strikes the bottom row, you will need to adjust the position to the brick's top row in order to erase one complete brick. The test is a very simple Boolean function in line 320. For example if the ball's new vertical position, NY=9, then $NY/2 \neq INT(NY/2)$ would reduce to $9/2 \neq 4$ which is true. We would then decrement NY to an even number in order to erase the complete block. The top left corner of the 2 pixel by 5 pixel brick is C,NY. Five pixels are erased from C,NY to C+4,NY in each of its two rows.

The brick's score depends on its playfield color. SCORE = SCORE + SCORE(C), where C is the value returned by the locate function. The yellow (playfield #1) bricks at the top are worth ten points, the green (playfield #2) bricks in the middle are worth five points, and the blue (playfield #3) bricks at the bottom are worth only three points.

The ball's vertical direction of travel reverses upon collision with a brick. It continues in the horizontal direction until it reaches either the left or right playfield boundary at BX=0 or BX=79. It reverses direction there so that DX = -DX. If the ball reaches the top of the playfield at BY = 0, it will reverse its vertical direction. But if the ball reaches the bottom it is lost and we begin again with a new ball. The game will end when we have run out of either bricks or balls.

```
5 REM BREAKOUT GAME - BY DAN PINAL
10 DIM C(3),SCORE(3)
20 C(0)=-0.5:C(1)=-1:C(2)=1:C(3)=0.5:SCORE(0)=0:SCORE(1)=10:SCORE(2)=5:SCORE(3)=3
30 GRAPHICS 5:POKE 752,1
40 FOR L1=0 TO 3:COLOR L1
50 FOR L2=0 TO 2 STEP 2
60 PLOT 0,L2+L1*4:DRAWTO 79,L2+L1*4
70 PLOT 0,L2+L1*4+1:DRAWTO 79,L2+L1*4+1
80 NEXT L2:NEXT L1
90 BALL=5:BRICKS=96
100 P=38:BX=INT(80*RND(0)):BY=17:DX=C(INT(4*RND(0))):DY=1:TX=BX
110 COLOR 1:PLOT BX,BY:GOSUB 1100
120 D=0:S=STICK(0):IF S>8 AND S<12 THEN D=-1
130 IF S>4 AND S<8 THEN D=1
140 IF  NOT STRIG(0) THEN D=D*3
150 P=P+D*((P+D)>0 AND P+D<76)
160 COLOR 0:PLOT PX,36:DRAWTO PX+3,36
170 COLOR 1:PX=P:PLOT PX,36:DRAWTO PX+3,36
180 IF STRIG(0)=0 AND STICK(0)=15 THEN 200
190 GOTO 120
200 D=0:S=STICK(0):IF S>8 AND S<12 THEN D=-1
```

Breakout Game



```
210 IF S>4 AND S<8 THEN D=1
220 IF  NOT STRIG(0) THEN D=D*3
230 P=P+D*((P+D)>0 AND P+D<76)
235 COLOR 0:PLOT TX,BY:PLOT PX,36:DRAWTO PX+3,36
240 COLOR 1:PX=P:PLOT PX,36:DRAWTO PX+3,36
250 BX=BX+DX:BY=BY+DY:TX=INT(BX):NX=INT(BX+DX*(BX+DX<80 AND BX+DX>=0)):NY=BY+DY*
    (BY+DY>=0)
260 LOCATE NX,NY,C:IF  NOT C THEN 350
270 IF BY<35 THEN 300
280 C=NX-PX:DY=-1:DX=C(C):GOSUB 1000:GOTO 350
300 SCORE=SCORE+SCORE(C):BRICKS=BRICKS-1:GOSUB 1000
310 C=INT(NX/5)*5:COLOR 0:DY=-DY
320 IF NY/2<>INT(NY/2) THEN NY=NY-1
330 FOR L1=0 TO 1:PLOT C,NY+L1:DRAWTO C+4,NY+L1:NEXT L1
340 GOSUB 1100
350 IF BX<1 OR BX>78 THEN DX=-DX:GOSUB 1000
360 IF BY=0 THEN DY=1:GOSUB 1000
370 IF BY=39 THEN 400
380 COLOR 1:PLOT TX,BY
390 IF BRICKS THEN 200
395 ? "             PERFECT":GOSUB 1300:GOTO 420
400 GOSUB 1200:BALL=BALL-1:IF BALL THEN 100
410 GOSUB 1100:? "             GAME OVER"
420 IF PEEK(53279)=6 THEN RUN
430 GOTO 420
1000 FOR L1=15 TO 0 STEP -1
1010 SOUND 0,30,10,L1
1020 NEXT L1:RETURN
1100 ? "}BALLS: ";BALL;"                SCORE: ";SCORE:RETURN
1200 FOR L1=80 TO 255:SOUND 0,L1,10,L1/8:NEXT L1:RETURN
1300 FOR L1=100 TO 10 STEP -5
1310 FOR L2=L1 TO 0 STEP -4
1320 SOUND 0,L2,10,6:NEXT L2:NEXT L1:SOUND 0,0,0,0:RETURN
```

Breakout Game (Assembly Language)

The "Breakout" game is quite easy to translate into Assembly language once you understand how BASIC handles its graphics commands. The Operating System (OS) implements each of these commands through the CIO (Central Input/Output) subroutine located at $E456. When a program calls the OS through this location, the OS expects to be given the address of a properly formatted IOCB (Input Output Control Block). There are eight of these, each sixteen bytes long. These are located from $340 to $3BF. The appropriate IOCB number times 16 is passed to the subroutine in the X-register. The full details of how the internals actually work are really not important, especially to the beginning Assembly language programmer. Let's just say that we developed a set of graphics subroutines that mirror their BASIC language counterpart. We have commented on each of these in the listing for anyone who would like to study them.

Graphics Commands

The five graphics commands that we need for our game are: GRAPHICS #, POSITION H,V; PLOT H,V; DRAWTO H,V; and LOCATE H,V,Color. We set up

each by inputting certain parameters into the Accumulator, X-register, and Y-register. Once you've set up the registers you need only JSR to that subroutine. The table below shows what you need to input into each of the registers.

Function	Accumulator	X-register	Y-register
GRAPHICS	Mode #	—————	—————
POSITION	Vertical	Horizontal High byte	Horizontal Low byte
PLOT	Vertical	Horizontal High byte	Horizontal Low byte
DRAWTO	Vertical	Horizontal High byte	Horizontal Low byte
LOCATE	Vertical Has color value on return.	Horizontal High byte	Horizontal Low byte

For example, if we wish to set up a Graphics 5 screen and draw a blue (playfield #3 default color) line from 10,15 to 30,15 our program would be as follows:

```
LDA #$05    ;GRAPHICS 5 SCREEN
JSR GRAPHICS
LDA #$03    ;PLAYFIELD #3
STA COLOR
LDA #$0F    ;VERTICAL=15
LDX #$00    ;HORIZONTAL HIGH BYTE
LDY #$0A    ;HORIZONTAL LOW BYTE
JSR PLOT    ;PLOT PIXEL
LDA #$0F    ;VERTICAL=15
LDX #$00    ;HORIZONTAL HIGH BYTE
LDY #$1E    ;HORIZONTAL LOW BYTE
JSR DRAWTO  ;DRAW LINE
```

Breakout Game

Once you understand the simplicity of duplicating the BASIC graphics statements in Assembly language you can proceed with developing the game.

The "Breakout" game is a very close translation of the BASIC version with a few subtle differences. One of the problems in working with Assembly language is that all numbers are whole integer numbers. In the BASIC version the ball's horizontal direction (DX) became +½ or -½ when it hit the inner portion of the paddle. Since incrementing the ball's position by +½ would be impossible in Assembly language, DX and BALLX, a temporary value for the ball's horizontal position, are doubled in value. If we then divide BALLX by two before plotting the ball's true position, TX,

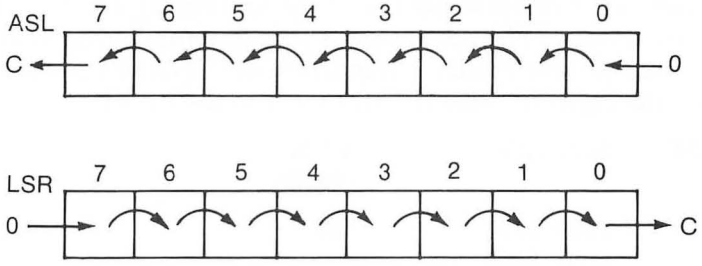
the fractional part, will vanish. In essence the ball will move horizontally every other frame.

BALLX = BALLX + DX (doubled values)
TX = BALLX / 2

ASL and LSR Instructions

Multiplication and division by powers of two is easy in Machine language. The mnemonic ASL is used for multiplication by two. The Arithmetic Shift Left (ASL) instruction shifts all of the bits in the Accumulator one position to the left. Thus, bit 0 is shifted into bit 1, bit 1 into bit 2, etc. Bit 7 is shifted into the carry bit so that you can use the BCC and BCS instructions to test for overflows. For example, if only bit 2 was on (4 decimal) and we did an ASL, the bit would be shifted to bit 3 (8 decimal). Thus, it is easy to multiply by powers of two by performing repeated ASL instructions.

Conversely, division is performed by the Logical Shift Right (LSR) instruction. Bits are shifted to the right and the bit 0 is shifted into the carry. This is equivalent to dividing by two with loss of the fractional part.




```
LDA #$05     ;LOAD ACCUMULATOR WITH 5
LSR          ;DIVIDE BY 2
STA $4000    ;VALUE STORED IN $4000 IS 2
```

Ball's Direction After Paddle Collision

The table of directional values for the four possible collision positions with the paddle are stored in VX. The two negative values in the table are stored in their two's complement form because it is easier to add two positive numbers rather than to test for a negative number and subtract.

	0th	1st	2nd	3rd
VX	$FE	$FF	$01	$02

For example, #$FE (-2)+ #$03 = #$01. The offset position from the paddle's left edge is placed in the X register to get the new horizontal velocity.

```
LDA TX      ;COMPARE PADDLE HORIZ. WITH BALL HORIZ.
SBC PX      ;DIFFERENCE
TAX
LDA VX,X    ;FETCH VELOCITY VALUE FROM TABLE
STA DX      ;THIS IS DOUBLED VALUE
```

We calculate the ball's new position as follows;

```
CLC
LDA BALLX   ;OLD BALL POSITION (DOUBLED)
ADC DX      ;NEW HORIZ. VELOCITY DOUBLED
STA BALLX
LSR         ;DIVIDE BY 2
STA TX      ;BALL'S TRUE HORIZ. POSITION
```

Scorekeeping

The scorekeeping routine also deserves an explanation. It differs substantially from the routines used in the other Machine language games in this book. It takes advantage of the 6502's ability to work in a numbering system called Binary Coded Decimal, or BCD. This system uses the lower four bits or low-order nibble to represent the low-order decimal digit, and the high-order nibble to represent the high-order decimal digit. The advantage is that the numbering system resembles decimal. The disadvantage is that it requires some advanced programming technique to isolate the digits in order to print them to the screen.

DECIMAL	BINARY	HEX (BCD)
07	0000 0111	$07
10	0001 0000	$10
16	0001 0110	$16
42	0100 0010	$42

To get to this mode you must set the decimal flag with a SED (Set Decimal Mode) command. It remains in effect until it is cleared by a CLD (Clear Decimal Mode) command.

A pair of bytes, SCORE and SCORE+1 are used to store the four score digits. These are updated by adding POINTS,X to SCORE+1 each time a brick is removed. The X-register contains the color value of the block hit so that we need only index into a table of point values. We didn't clear the carry when we added #$00 to SCORE (highbyte). However, it there was an overflow in SCORE+1 (low byte) during the first addition, the carry would be included in the resulting value in SCORE. Each of the four nibbles must be separated, translated into an internal character #, and finally placed into the appropriate position in the text window. The byte's high nibble is first shifted to the low nibble by four successive LSR instructions and then translated into an internal character number. Digits in the internal character set begin at #$10. Internal character #16 decimal = 0, 17 decimal = 1, etc. The ORA #$10

instruction, which combines the individual bits in its operand with those in the Accumulator, is just a fancy way of adding $10 to the value of our digit. The value of the low nibble is isolated by ANDing it with #$0F. It is then ORed with #$10 to obtain the internal character and stored in the next screen position. We have effectively stored the thousands and hundreds digits in the screen window. The code loops back again to obtain the value for the two nibbles in SCORE+1. These contain the tens and units digits. All of the store operations are done using indirect indexed addressing of the form STA(WINDOW),Y. We will discuss this at greater length in later chapters. Meanwhile, it allows us to index rapidly into a memory area whose two-byte address is stored in zero page.

If you're confused or lost at this point, don't worry. Just read on. Our intention was merely to show how a simple game like "Breakout" could be translated into Assembly language using graphics subroutines. It is not necessary to understand all of the details but to be able to roughly follow the code as it pertains to the game's flow chart. Many of the subtle tricks we mentioned in the previous discussion we will discuss in much greater detail in subsequent chapters.

```
                00020 *BREAKOUT GAME - BY DAN PINAL
                00030 *ZERO PAGE EQUATES
00F0:           00040 WINDOW    .EQ $F0
00F2:           00050 MSG       .EQ $F2
00F4:           00060 LINE2     .EQ $F4
00F6:           00070 PO        .EQ $F6
                00080 *MISC EQUATES
0014:           00090 RTCLOC    .EQ $14
004D:           00100 ATRACT    .EQ $4D
0230:           00110 SDLSTL    .EQ $230
0278:           00120 STICK0    .EQ $278
0284:           00130 STRIG0    .EQ $284
D200:           00140 AUDF1     .EQ $D200    ; PITCH 1
D201:           00150 AUDC1     .EQ $D201    ; DISTORTION/VOLUME 1
D20A:           00160 RANDOM    .EQ $D20A
E45C:           00170 SETVBV    .EQ $E45C    ; SET VBLANK ROUTINE
E462:           00180 XITVBV    .EQ $E462    ; DEFFERED VBLANK EXIT
D01F:           00190 CONSOL    .EQ $D01F
                00200 * EQUATES FOR CIO
E456:           00210 CIOV      .EQ $E456    ; ENTRY VECTOR
                00220 * COMMANDS *
0003:           00230 OPEN      .EQ $03      ; OPEN FOR INPUT/OUTPUT
0005:           00240 GETREC    .EQ $05      ; GET RECORD
0007:           00250 GETCHR    .EQ $07      ; GET CHARACTER(S)
0009:           00260 PUTREC    .EQ $09      ; PUT RECORD
000B:           00270 PUTCHR    .EQ $0B      ; PUT CHARACTER(S)
000C:           00280 CLOSE     .EQ $0C      ; CLOSE DEVICE
000D:           00290 STATIS    .EQ $0D      ; STATUS REQUEST
000E:           00300 SPECIL    .EQ $0E      ; BEGINNING OF SPECIAL ENTRY COMMANDS
                00310 *
                00320 * SPECIAL COMMANDS *
                00330 *
0011:           00340 DRAWLN    .EQ $11      ; DRAW LINE
0012:           00350 FILLIN    .EQ $12      ; DRAW LINE WITH RIGHT FILL
                00360 *
                00370 * AUX1 VALUES *
                00380 *
```

```
0001:           00390 APPEND   .EQ $01     ; OPEN FOR WRITE APPEND
0004:           00400 OPNIN    .EQ $04     ; OPEN FOR INPUT
0008:           00410 OPNOT    .EQ $08     ; OPEN FOR OUTPUT
000C:           00420 OPNINO   .EQ OPNIN+OPNOT   ; OPEN FOR INPUT AND OUTPUT
0010:           00430 MXDMOD   .EQ $10     ; OPEN FOR MIXED MODE (E:,S:)
0020:           00440 INSCLR   .EQ $20     ; OPEN WITHOUT CLEARING SCREEN (E:,S:)
                00450 *
                00460 * O.S. RAM EQUATES
                00470 *I/O CONTROL BLOCK EQUATES
0054:           00480 ROWCRS   .EQ $54     ; CURRENT GRAPHICS CURSOR ROW
0055:           00490 COLCRS   .EQ $55     ; & $56. LSB MSB OF CURRENT GRAPHICS CURSOR COLUMN
02FB:           00500 ATACHR   .EQ $2FB    ; LAST GRAPHICS CHARACTER READ OR WRITTEN
                00510 *
0340:           00520 ICHID    .EQ $340    ; HANDLER INDEX SET BY O.S.
0341:           00530 ICDNO    .EQ $341    ; DEVICE # AS IN D1:,D2: ETC
0342:           00540 ICCOM    .EQ $342    ; COMMAND
0343:           00550 ICSTA    .EQ $343    ; STATUS RETURNED
0344:           00560 ICBAL    .EQ $344    ; BUFFER ADDRESS LO
0345:           00570 ICBAH    .EQ $345    ; BUFFER ADDRESS HI
0346:           00580 ICPTL    .EQ $346    ; PUT ONE BYTE VECTOR LO
0347:           00590 ICPTH    .EQ $347    ; PUT ONE BYTE VECTOR HI
0348:           00600 ICBLL    .EQ $348    ; BUFFER LENGTH LO
0349:           00610 ICBLH    .EQ $349    ; BUFFER LENGTH HI
034A:           00620 ICAUX1   .EQ $34A    ; AUX1
034B:           00630 ICAUX2   .EQ $34B    ; AUX2
                00640 *
4000: 4C 30 42  00650          JMP START   ; SKIP PAST SUBROUTINES
                00660 * GRAPHICS CALLS TO O.S. *
                00670 *
                00680 ; GRAPHICS
                00690 ; ACC=GR. MODE
4003: 48        00700 GRAPHICS PHA         ; SAVE MODE
4004: A2 60     00710          LDX #$60    ; IOCB #6 FOR GRAPHICS
4006: A9 0C     00720          LDA #CLOSE  ;CLOSE #6 FOR SAFETY (CAN'T OPEN AN ALREADY
                                            OPEN DEVICE)
4008: 9D 42 03  00730          STA ICCOM,X
400B: 20 56 E4  00740          JSR CIOV
400E: A9 03     00750          LDA #OPEN
4010: 9D 42 03  00760          STA ICCOM,X
4013: A9 AC     00770          LDA #FNAME  ; FILENAME IS S:
4015: 9D 44 03  00780          STA ICBAL,X
4018: A9 41     00790          LDA /FNAME
401A: 9D 45 03  00800          STA ICBAH,X
401D: 68        00810          PLA         ; GR. MODE
401E: 9D 4B 03  00820          STA ICAUX2,X
4021: 29 F0     00830          AND #$F0    ; KEEP UPPER NIBBLE FOR FULLSCREEN
                00835 *                    ;& NO SCREEN ERASE FLAGS(+16 & +32)
                00840 * O.S. FULL SCREEN FLAG IS THE OPPOSITE OF THE WAY BASIC USES IT.
                00845 *i.e. +16 = SPLIT SCREEN NOT FULL SCREEN
4023: 49 10     00850          EOR #$10    ; FLIP-FLOP IT LIKE BASIC
4025: 09 0C     00860          ORA #OPNINO ; OPEN S: FOR READING & WRITING
4027: 9D 4A 03  00870          STA ICAUX1,X
402A: 4C 56 E4  00880          JMP CIOV    ; GO DO IT IT WILL RTS FROM THERE
                00890 ;
                00900 ; POSITION
                00910 ; EVEN THOUGH THIS IS SIMPLE IT IS HERE AS A SUBROUTINE
                00920 ; BECAUSE IT IS USED OFTEN BY OTHER ROUTINES
                00930 ; ACC. = VERT.
                00940 ; X = HORIZ. HI : Y = HORIZ. LO
                00950 POSITION
402D: 84 55     00960          STY COLCRS  ; CURSOR COLUMN LO
402F: 86 56     00970          STX COLCRS+1 ;CURSOR COLUMN HI
```

```
4031: 85 54    00980          STA ROWCRS  ; CURSOR ROW
4033: 60       00990          RTS
               01000 ;
               01010 ; PLOT
               01020 ; ENTRY REG. ARE SAME AS POSITION
               01030 ; WILL PLOT A POINT USING VALUE IN COLOR
               01040 ;
4034: 20 2D 40 01050 PLOT     JSR POSITION
4037: A2 60    01060          LDX #$60    ; IOCB #6
4039: A9 0B    01070          LDA #PUTCHR
403B: 9D 42 03 01080          STA ICCOM,X
403E: A9 00    01090          LDA #$0     ; 0 BUFFER LENGTH
4040: 9D 48 03 01100          STA ICBLL,X
4043: 9D 49 03 01110          STA ICBLH,X
4046: AD 15 42 01120          LDA COLOR   ; CIO WILL DEFAULT TO VALUE IN ACC
4049: 4C 56 E4 01130          JMP CIOV    ; & SEND IT TO SCREEN (IT WILL RTS TO PROGRAM FROM CIO
               01140 ;
               01150 ; DRAWTO
               01160 ; ENTRY SETUP SAME AS POSITION
404C: 20 2D 40 01170 DRAWTO   JSR POSITION
404F: AD 15 42 01180          LDA COLOR
4052: 8D FB 02 01190          STA ATACHR  ; TELL O.S.
4055: A2 60    01200          LDX #$60    ; IOCB #6
4057: A9 11    01210          LDA #DRAWLN ; DRAWTO HAS ITS OWN SPECIAL COMMAND
4059: 9D 42 03 01220          STA ICCOM,X
405C: A9 0C    01230          LDA #OPNINO
405E: 9D 4A 03 01240          STA ICAUX1,X; MAKE SURE O.S. KNOWS WHAT IT IS SUPPOSED TO DO
4061: A9 00    01250          LDA #$0
4063: 9D 4B 03 01260          STA ICAUX2,X
4066: 4C 56 E4 01270          JMP CIOV    ; GO DO IT & RTS TO PROGRAM FROM THERE
               01280 ;
               01290 ; LOCATE
               01300 ; SETUP SAME AS POSITION
               01310 ; ACC. & COLOR HAVE VALUE LOCATED
4069: 20 2D 40 01320 LOCATE   JSR POSITION
406C: A2 60    01330          LDX #$60    ; IOCB #6
406E: A9 07    01340          LDA #GETCHR
4070: 9D 42 03 01350          STA ICCOM,X
4073: A9 00    01360          LDA #$0     ; 0 OUT BUFFER LENGTH
4075: 9D 48 03 01370          STA ICBLL,X
4078: 9D 49 03 01380          STA ICBLH,X
407B: 20 56 E4 01390          JSR CIOV    ; THIS WORKS EXACTLY THE OPPOSITE OF PLOT
407E: 8D 15 42 01400          STA COLOR
4081: 60       01410          RTS
               01420 ;
               01430 * MISC. ROUTINES
               01440 * 8 BIT DIVISION
               01450 ; X= NUM TO DIVIDE BY
               01460 ; A= LOBYTE OF NUM TO DIVIDE INTO
               01470 ; Y= HIBYTE OF NUM TO DIVIDE INTO
               01480 ; RESULT & REMAIN HAVE ANSWER ON EXIT
               01490 ; A=REMAINDER
4082: 8D 16 42 01500 DIVIDE   STA RESULT
4085: 8E 18 42 01510          STX DIVISOR
4088: 98       01520          TYA
4089: A2 08    01530          LDX #$08
408B: 0E 16 42 01540 .1       ASL RESULT
408E: 2A       01550          ROL
408F: CD 18 42 01560          CMP DIVISOR
4092: 90 06    01570          BCC .2
4094: ED 18 42 01580          SBC DIVISOR
4097: EE 16 42 01590          INC RESULT
```

```
409A: CA       01600 .2       DEX
409B: D0 EE    01610          BNE .1
409D: 8D 17 42 01620          STA REMAIN
40A0: 60       01630          RTS
               01640 ;
               01650 ; 8 BIT MULTIPLY
               01660 ; MULTIPLIES ACC BY X REG.
               01670 ; STORES RESULT (L,H) IN RESULT & RESULT+1
40A1: 8D 19 42 01680 MULTIPLY STA M1      ; MULTIPLIER
40A4: 8E 1A 42 01690          STX M2      ; MULTIPLICAND
40A7: A9 00    01700          LDA #$0
40A9: 8D 16 42 01710          STA RESULT
40AC: 8D 17 42 01720          STA RESULT+1
40AF: A2 08    01730          LDX #$08
40B1: 0E 16 42 01740 .1       ASL RESULT
40B4: 2E 17 42 01750          ROL RESULT+1
40B7: 0E 1A 42 01760          ASL M2
40BA: 90 0F    01770          BCC .2
40BC: 18       01780          CLC
40BD: AD 16 42 01790          LDA RESULT
40C0: 6D 19 42 01800          ADC M1
40C3: 8D 16 42 01810          STA RESULT
40C6: 90 03    01820          BCC .2
40C8: EE 17 42 01830          INC RESULT+1
40CB: CA       01840 .2       DEX
40CC: D0 E3    01850          BNE .1
40CE: 60       01860          RTS
               01870 ;
40CF: A9 01    01880 NEWBALL  LDA #$01    ; BALL COLOR
40D1: 8D 15 42 01890          STA COLOR
40D4: 18       01900          CLC
40D5: AD 0A D2 01910          LDA RANDOM
40D8: 29 7F    01920          AND #$7F ; 0-128
40DA: 69 10    01930          ADC #$10 ; 16-144
40DC: 8D 20 42 01940          STA BALLX ; BALL HORIZ. POS. IS DOUBLED SO IT CAN BE MOVED IN HALF INC
40DF: 4A       01950          LSR       ; /2 FOR ACTUAL BALL POSITION
40E0: 8D 25 42 01960          STA OLDX
40E3: A8       01970          TAY
40E4: A2 00    01980          LDX #$00
40E6: A9 11    01990          LDA #$11    ; 17
40E8: 8D 21 42 02000          STA BALLY
40EB: 8D 26 42 02010          STA OLDY
40EE: 20 34 40 02020          JSR PLOT    ; PUT NEW BALL STARTING POS. ON SCREEN
40F1: A9 01    02030          LDA #$01
40F3: 8D 23 42 02040          STA DY      ; SET FOR DOWN
40F6: AD 0A D2 02050          LDA RANDOM
40F9: 29 03    02060          AND #$03    ; 0-3
40FB: AA       02070          TAX
40FC: BD 0D 42 02080          LDA VX,X    ; PICK RANDOM HORIZ. VECTOR
40FF: 8D 22 42 02090          STA DX      ; SAVE IT
4102: A9 26    02100          LDA #$26    ; 38
4104: 8D 27 42 02110          STA PADDLE  ; REPOSITION TO MIDDLE OF SCREEN
4107: 20 17 41 02120 .1       JSR DOPADDLE
410A: AD 78 02 02130          LDA STICKO
410D: C9 0F    02140          CMP #$0F    ; STICK CENTER?
410F: D0 F6    02150          BNE .1      ; WAIT TILL IT IS.
4111: AD 84 02 02160          LDA STRIGO  ; BUTTON PRESSSED?
4114: D0 F1    02170          BNE .1      ; WAIT TILL TRIGGER PRESSED
4116: 60       02180          RTS
               02190 *
4117: AE 78 02 02200 DOPADDLE LDX STICKO  ; GET STICK VALUE
411A: AD 84 02 02210          LDA STRIGO  ; BUTTON PRESSED
```

```
411D: D0 06    02220          BNE .1      ; USE NORMAL SPEED PADDLE MOVEMENT
411F: BD FD 41 02230          LDA FHOFFS,X ; TRIPLE SPEED TABLE
4122: 4C 28 41 02240          JMP .2
4125: BD ED 41 02250 .1       LDA HOFFS,X ; NORMAL SPEED TABLE
4128: A2 00    02260 .2       LDX #$00    ; PADDLE MINIMUM
412A: A0 4C    02270          LDY #$4C    ; 76 PADDLE MAXIMUM
 12C: 18       02280          CLC
412D: 6D 27 42 02290          ADC PADDLE
4130: 10 01    02300          BPL .3
4132: 8A       02310          TXA         ; PADDLE H. CAN'T GO LESS THAN 0 SO SET IT TO 0
4133: C9 4D    02320 .3       CMP #$4D    ; 77 GONE PAST UPPER LIMIT
4135: 90 01    02330          BCC .4      ; OK IF <
4137: 98       02340          TYA
4138: 8D 27 42 02350 .4       STA PADDLE
               02360 * NOW ERASE OLD PADDLE & DRAW NEW
413B: A9 00    02370          LDA #$00
413D: 8D 15 42 02380          STA COLOR
4140: 20 52 41 02390          JSR DRPADDLE
4143: AD 27 42 02400          LDA PADDLE
4146: 8D 28 42 02410          STA PX
4149: A9 01    02420          LDA #$01
414B: 8D 15 42 02430          STA COLOR
414E: 20 52 41 02440          JSR DRPADDLE,
4151: 60       02450          RTS
               02460 *
               02470 * DRAW PADDLE
4152: A9 24    02480 DRPADDLE LDA #$24    ; 36TH ROW
4154: AC 28 42 02490          LDY PX      ; PADDLE H.
4157: A2 00    02500          LDX #$00    ; HI BYTE OF PADDLE H. ALWAYS 0 UNLESS GR. 8
4159: 20 34 40 02510          JSR PLOT
415C: 18       02520          CLC
415D: AD 28 42 02530          LDA PX
4160: 69 03    02540          ADC #$03    ; END OF PADDLE IS PADDLE H. + 3
4162: A8       02550          TAY         ; SET UP FOR DRAWTO CALL
4163: A2 00    02560          LDX #$00
4165: A9 24    02570          LDA #$24
4167: 20 4C 40 02580          JSR DRAWTO
416A: 60       02590          RTS
               02600 *
               02610 *EBRICK
               02620 * ERASES BRICK ONE LINE AT A TIME
416B: A2 00    02630 EBRICK   LDX #$00
416D: 8E 15 42 02640          STX COLOR
4170: AD 2C 42 02650          LDA BRICKY
4173: AC 2B 42 02660          LDY BRICKX
4176: 20 34 40 02670          JSR PLOT
4179: 18       02680          CLC
417A: AD 2B 42 02690          LDA BRICKX
417D: 69 04    02700          ADC #$04
417F: A8       02710          TAY
4180: A2 00    02720          LDX #$00
4182: AD 2C 42 02730          LDA BRICKY
4185: 20 4C 40 02740          JSR DRAWTO
4188: 60       02750          RTS
               02760 *
4189: A5 14    02770 JDELAY   LDA RTCLOC
418B: C5 14    02780 .1       CMP RTCLOC
418D: F0 FC    02790          BEQ .1
418F: CA       02800          DEX
4190: D0 F7    02810          BNE JDELAY
4192: 60       02820          RTS
               02830 *
```

```
                02840 * VBI ROUTINE IS USED SO SOUND DOES NOT INTERFERE WITH PLAY
                02850 VBI
4193: D8        02860           CLD
4194: AD 2F 42  02870           LDA TIMER    ; SOUND TO DO
4197: F0 10     02880           BEQ XVBI     ; NO. LEAVE
4199: CE 2F 42  02890           DEC TIMER    ; COUNTDOWN
419C: AD 2F 42  02900           LDA TIMER    ; USE TIMER AS VOLUME
419F: 09 A0     02910           ORA #$A0     ; DISTORTION 10
41A1: 8D 01 D2  02920           STA AUDC1    ; TELL POKEY
41A4: A9 20     02930           LDA #$20
41A6: 8D 00 D2  02940           STA AUDF1
                02950 XVBI
41A9: 4C 62 E4  02960           JMP XITVBV
                02970 *
                02980 * PROGRAM DATA
                02990 *
41AC: 53 3A     03000 FNAME     .AS "S:"
41AE: 9B        03010           .HS 9B
41AF: 27 21 2D
41B2: 25 00 2F
41B5: 36 25 32  03020 MSG0      .AT "GAME OVER"
41B8: 00 30 25
41BB: 32 26 25
41BE: 23 34 01  03030 MSG1      .AT " PERFECT!"
41C1: AF 41 B8
41C4: 41        03040 MSGTAB    .DA MSG0,MSG1
41C5: 00 00 22
41C8: 21 2C 2C
41CB: 1A 00 10
41CE: 00 00 00
41D1: 00 00 00
41D4: 00 00 00
41D7: 00 00     03050 TOPLINE   .AT "  BALL: 0           "
41D9: 00 00 00
41DC: 00 00 00
41DF: 00 00 33
41E2: 23 2F 32
41E5: 25 1A 00
41E8: 10 10 10
41EB: 10 00     03060           .AT "        SCORE: 0000 "
41ED: 00 00 00
41F0: 00 00 01
41F3: 01 01     03070 HOFFS     .HS 0000000000010101
41F5: 00 FF FF
41F8: FF 00 00
41FB: 00 00     03080           .HS 00FFFFFF00000000
41FD: 00 00 00
4200: 00 00 03
4203: 03 03     03090 FHOFFS    .HS 0000000000030303
4205: 00 FD FD
4208: FD 00 00
420B: 00 00     03100           .HS 00FDFDFD00000000
420D: FE FF 01
4210: 02        03110 VX        .HS FEFF0102
4211: 00 10 05
4214: 03        03120 POINTS    .HS 00100503
4215:           03130 COLOR     .BS 1
4216:           03140 RESULT    .BS 1        ; RESULT & REMAIN MUST STAY
4217:           03150 REMAIN    .BS 1        ; NEXT TO EACH OTHER
4218:           03160 DIVISOR   .BS 1
4219:           03170 M1        .BS 1
421A:           03180 M2        .BS 1
```

```
421B:           03190 TEMP1    .BS 1
421C:           03200 TEMP2    .BS 1
421D:           03210 TEMP3    .BS 1
421E:           03220 TEMP4    .BS 1
421F:           03230 ROW      .BS 1
4220:           03240 BALLX    .BS 1       ;BALL'S X POSITION (DOUBLED)
4221:           03250 BALLY    .BS 1       ;BALL'S Y POSITION
4222:           03260 DX       .BS 1       ;DOUBLED HORIZONTAL DIRECTION VALUE
4223:           03270 DY       .BS 1       ;VERTICAL DIRECTION VALUE
4224:           03280 TX       .BS 1       ;TRUE HORIZONTAL DIRECTION VALUE
4225:           03290 OLDX     .BS 1       ;BALL'S OLD HORIZ POSITION
4226:           03300 OLDY     .BS 1       ;BALL'S OLD VERTICAL POSITION
4227:           03310 PADDLE   .BS 1       ;HORIZ PADDLE POSITION
4228:           03320 PX       .BS 1       ;NEW HORIZ PADDLE POSITION
4229:           03330 BALLS    .BS 1
422A:           03340 BRICKS   .BS 1
422B:           03350 BRICKX   .BS 1       ;HORIZ POSITION OF BRICK TO BE REMOVED
422C:           03360 BRICKY   .BS 1       ;VERTICAL POSITION
422D:           03370 SCORE    .BS 2
422F:           03380 TIMER    .BS 1
                03390 ;
                03400 *BEGINNING OF MAIN PROGRAM
4230: A9 07     03410 START    LDA #$07    ; DEFFERED
4232: A0 93     03420          LDY #VBI
4234: A2 41     03430          LDX /VBI
4236: 20 5C E4  03440          JSR SETVBV  ; ENABLE SOUND ROUTINE
4239: A9 05     03450          LDA #$05    ; START WITH GRAPHICS 5
423B: 20 03 40  03460          JSR GRAPHICS
423E: AD 30 02  03470          LDA SDLSTL  ; FIND DISPLAY LIST
4241: 85 F6     03480          STA P0
4243: AD 31 02  03490          LDA SDLSTL+1
4246: 85 F7     03500          STA P0+1
4248: A0 2F     03510          LDY #$2F
424A: B1 F6     03520          LDA (P0),Y  ; FIND OUT WHERE IT PUT TEXT WINDOW
424C: 85 F1     03530          STA WINDOW+1
424E: 85 F5     03540          STA LINE2+1
4250: 88        03550          DEY
4251: B1 F6     03560          LDA (P0),Y
4253: 85 F0     03570          STA WINDOW
4255: 18        03580          CLC
4256: 69 38     03590          ADC #$38    ; 56 ALMOST HALFWAY INTO 2ND LINE OF TEXT WINDOW
4258: 85 F4     03600          STA LINE2
425A: 90 02     03610          BLT .1
425C: E6 F5     03620          INC LINE2+1
425E: A0 27     03630 .1       LDY #$27    ; 39
4260: B9 C5 41  03640 .2       LDA TOPLINE,Y     ; BLOCK MOVE THE TOPLINE INTO TEXT WINDOW
4263: 91 F0     03650          STA (WINDOW),Y
4265: 88        03660          DEY
4266: 10 F8     03670          BPL .2
                03680 * NOW DRAW BRICKS
4268: A9 03     03690 DOBRICKS LDA #$03
426A: 8D 1B 42  03700          STA TEMP1
426D: AD 1B 42  03710 .1       LDA TEMP1
4270: 8D 15 42  03720          STA COLOR
4273: A9 02     03730          LDA #$02
4275: 8D 1C 42  03740          STA TEMP2
4278: AD 1B 42  03750 .2       LDA TEMP1
427B: 0A        03760          ASL         ; X 2
427C: 0A        03770          ASL         ; X 4
427D: 18        03780          CLC
427E: 6D 1C 42  03790          ADC TEMP2
4281: 8D 1F 42  03800          STA ROW
```

```
4284: A9 02    03810          LDA #$02
4286: 8D 1D 42 03820          STA TEMP3
4289: AD 1F 42 03830 .3       LDA ROW
428C: A2 00    03840          LDX #$00
428E: A0 00    03850          LDY #$00
4290: 20 34 40 03860          JSR PLOT     ; PLOT 0,ROW
4293: AD 1F 42 03870          LDA ROW
4296: A2 00    03880          LDX #$0
4298: A0 4F    03890          LDY #$4F     ; 79
429A: 20 4C 40 03900          JSR DRAWTO   ; DRAWTO 79,ROW
429D: EE 1F 42 03910          INC ROW
42A0: CE 1D 42 03920          DEC TEMP3
42A3: D0 E4    03930          BNE .3       ; TWO ROWS PER BRICK
42A5: CE 1C 42 03940          DEC TEMP2
42A8: CE 1C 42 03950          DEC TEMP2
42AB: 10 CB    03960          BPL .2       ; TWO ROWS OF BRICKS PER COLOR
42AD: CE 1B 42 03970          DEC TEMP1
42B0: D0 BB    03980          BNE .1       ; 3 COLORS OF BRICKS
42B2: A9 01    03990          LDA #$01
42B4: 8D 29 42 04000          STA BALLS
42B7: 09 10    04010          ORA #$10     ; MAKE INTERNERNAL CHR
42B9: A0 08    04020          LDY #$08
42BB: 91 F0    04030          STA (WINDOW),Y
42BD: A9 60    04040          LDA #$60     ; 96  6 ROWS OF 16 BRICKS
42BF: 8D 2A 42 04050          STA BRICKS
42C2: A9 00    04060          LDA #$00
42C4: 8D 2D 42 04070          STA SCORE
42C7: 8D 2E 42 04080          STA SCORE+1
               04090 * MAIN GAME LOOP BEGINS HERE
               04100 *
42CA: 20 CF 40 04110 GAME     JSR NEWBALL ; INITIALIZE NEW BALL DATA & WAIT FOR BUTTON PRESS
42CD: A2 08    04120 GAMELOOP LDX #$08
42CF: 86 4D    04130          STX ATRACT
42D1: 20 89 41 04140          JSR JDELAY
42D4: 20 17 41 04150          JSR DOPADDLE ; UPDATE PADDLE POS.
               04160 * GET NEW BALL POS.
42D7: 18       04170 DOBALL   CLC
42D8: AD 20 42 04180          LDA BALLX
42DB: 6D 22 42 04190          ADC DX       ; HORIZONTAL CHANGE. -2 TO +2
42DE: C9 F0    04200          CMP #$F0
42E0: 90 02    04210          BCC .0
42E2: A9 00    04220          LDA #$00
42E4: C9 A0    04230 .0       CMP #$A0
42E6: 90 02    04240          BCC .10
42E8: A9 9F    04250          LDA #$9F
42EA: 8D 20 42 04260 .10      STA BALLX
42ED: 4A       04270          LSR          ; TRUE H. POS.
42EE: 8D 24 42 04280          STA TX
42F1: F0 04    04290          BEQ .1
42F3: C9 4F    04300          CMP #$4F     ; 79  AT RIGHT EDGE?
42F5: 90 10    04310          BLT .2       ; OK IF <
42F7: CE 22 42 04320 .1       DEC DX
42FA: AD 22 42 04330          LDA DX
42FD: 49 FF    04340          EOR #$FF     ; REVERSE HORIZONTAL
42FF: 8D 22 42 04350          STA DX
4302: A9 10    04360          LDA #$10
4304: 8D 2F 42 04370          STA TIMER    ; PING!
4307: 18       04380 .2       CLC
4308: AD 21 42 04390          LDA BALLY
430B: 6D 23 42 04400          ADC DY
430E: 8D 21 42 04410          STA BALLY
```

```
4311: C9 02    04420          CMP #$02
4313: B0 0C    04430          BGE .3
4315: A9 01    04440          LDA #$01     ; AT TOP
4317: 8D 23 42 04450          STA DY       ; SET VERTICAL DIRECTION TO DOWN
431A: A9 10    04460          LDA #$10
431C: 8D 2F 42 04470          STA TIMER    ; PING!
431F: D0 07    04480          BNE HITCK    ; BRANCH ALWAYS
4321: C9 27    04490 .3       CMP #$27     ; PAST PADDLE?
4323: 90 03    04500          BLT HITCK    ; OK IF <
4325: 4C ED 43 04510          JMP ENDBALL
   ,           04520 ; ACC HAS V.
4328: A2 00    04530 HITCK    LDX #$00
432A: AC 24 42 04540          LDY TX
432D: 20 69 40 04550          JSR LOCATE   ; ACC. HAS COLOR VALUE OF PIXEL
4330: C9 00    04560          CMP #$00
4332: D0 03    04570          BNE .1
4334: 4C C4 43 04580          JMP XHITCK
4337: A0 10    04590 .1       LDY #$10
4339: 8C 2F 42 04600          STY TIMER    ; PING!
433C: AC 21 42 04610          LDY BALLY
433F: C0 20    04620          CPY #$20     ; HAS IT HIT PADDLE OR BRICKS?
4341: 90 15    04630          BCC .2       ; BRICKS
4343: AD 24 42 04640          LDA TX       ; COMPARE PADDLE H. TO BALL H.
4346: ED 28 42 04650          SBC PX       ; DIFFERENCE IS 0-3
4349: AA       04660          TAX          ; AND WILL DECIDE NEW HORIZ. VELOCITY
434A: BD 0D 42 04670          LDA VX,X
434D: 8D 22 42 04680          STA DX       ;STORE NEW HORIZ VELOCITY
4350: A9 FF    04690          LDA #$FF
4352: 8D 23 42 04700          STA DY       ; SET VERTICAL DIRECTION TO UP
4355: 4C C4 43 04710          JMP XHITCK   ; LEAVE
               04720 * ACC. STILL HAS COLOR VALUE HIT.  USE IT TO GET BRICK VALUE
4358: AA       04730 .2       TAX
4359: F8       04740          SED
435A: AD 2E 42 04750          LDA SCORE+1 ;LOAD TWO LOW DIGITS
435D: 7D 11 42 04760          ADC POINTS,X ;ADD POINT VALUE OF BRICK
4360: 8D 2E 42 04770          STA SCORE+1
4363: A9 00    04780          LDA #$00     ;WILL ALSO ADD CARRY BIT IF NECESSARY
4365: 6D 2D 42 04790          ADC SCORE
4368: 8D 2D 42 04800          STA SCORE
436B: D8       04810          CLD
               04820 * NOW PRINT NEW SCORE ON SCREEN
               04830 DOSCORE
436C: A2 00    04840          LDX #$00
436E: A0 23    04850          LDY #$23
4370: BD 2D 42 04860 .1       LDA SCORE,X
4373: 4A       04870          LSR          ; EACH BYTE HOLDS 2 NUMBERS
4374: 4A       04880          LSR          ; SHIFT UPPER NIBBLE OVER
4375: 4A       04890          LSR          ; AND DO IT
4376: 4A       04900          LSR          ; FIRST.
4377: 09 10    04910          ORA #$10     ; TRANSLATE NUMBER INTO INTERNAL CHARACTER
4379: 91 F0    04920          STA (WINDOW),Y ;STORE HIGHER DIGIT OF PAIR IN TEXT AREA
437B: C8       04930          INY          ;NEXT
437C: BD 2D 42 04940          LDA SCORE,X
437F: 29 0F    04950          AND #$0F     ; NOW DO LOWER NIBBLE
4381: 09 10    04960          ORA #$10     ; MAKE A CHR
4383: 91 F0    04970          STA (WINDOW),Y ;STORE LOWER DIGIT OF PAIR IN TEXT AREA
4385: C8       04980          INY
4386: E8       04990          INX
4387: E0 02    05000          CPX #$02     ; DONE BOTH BYTES? (ALL 4 SCORE DIGITS)
4389: D0 E5    05010          BNE .1
               05020 * NOW CALCULATE BRICK POSITION
```

```
                05030 * BRICK HORIZONTAL IS THE INTEGER OF TX/5
438B: AD 24 42  05040           LDA TX
438E: A0 00     05050           LDY #$00     ; TX HAS NO HIBYTE SO Y=0
4390: A2 05     05060           LDX #$05     ; # TO DIVIDE BY
4392: 20 82 40  05070           JSR DIVIDE
4395: AD 16 42  05080           LDA RESULT
4398: A2 05     05090           LDX #$05
439A: 20 A1 40  05100           JSR MULTIPLY
439D: AD 16 42  05110           LDA RESULT
43A0: 8D 2B 42  05120           STA BRICKX
43A3: AD 21 42  05130           LDA BALLY    ; BALL VERT.
43A6: 4A        05140           LSR          ; /2
43A7: 0A        05150           ASL          ; X2  THIS INSURES AN EVEN NUMBER
43A8: 8D 2C 42  05160           STA BRICKY   ; THE TOP OF ANY BRICK IS AN EVEN ROW.
43AB: 20 6B 41  05170           JSR EBRICK   ; ERASE TOP LINE OF BRICK
43AE: EE 2C 42  05180           INC BRICKY   ; BRICKS ARE 2 LINES HIGH
43B1: 20 6B 41  05190           JSR EBRICK   ; ERASE SECOND LINE
                05200 * BRICK IS GONE
                05210 * FLIP FLOP VERTICAL DIRECTION
43B4: CE 23 42  05220           DEC DY
43B7: AD 23 42  05230           LDA DY
43BA: 49 FF     05240           EOR #$FF
43BC: 8D 23 42  05250           STA DY
43BF: CE 2A 42  05260           DEC BRICKS   ; ONE LESS BRICK
43C2: F0 4A     05270           BEQ ENDGAME ; IF ALL ARE GONE END GAME
                05280 XHITCK
43C4: A9 00     05290           LDA #$0
43C6: 8D 15 42  05300           STA COLOR
43C9: AD 26 42  05310           LDA OLDY
43CC: A2 00     05320           LDX #$00
43CE: AC 25 42  05330           LDY OLDX
43D1: 20 34 40  05340           JSR PLOT     ; ERASE OLD BALL
43D4: A9 01     05350           LDA #$01
43D6: 3D 15 42  05360           STA COLOR
43D9: AD 21 42  05370           LDA BALLY
43DC: 8D 26 42  05380           STA OLDY
43DF: A2 00     05390           LDX #$00
43E1: AC 24 42  05400           LDY TX
43E4: 8C 25 42  05410           STY OLDX
43E7: 20 34 40  05420           JSR PLOT     ; PUT BALL ON IN NEW POSITION
43EA: 4C CD 42  05430           JMP GAMELOOP      ; DO IT AGAIN
43ED: A2 00     05440 ENDBALL   LDX #$00
43EF: 8E 15 42  05450           STX COLOR
43F2: AC 25 42  05460           LDY OLDX
43F5: AD 26 42  05470           LDA OLDY
43F8: 20 34 40  05480           JSR PLOT     ; ERASE BALL
43FB: EE 29 42  05490           INC BALLS
43FE: AD 29 42  05500           LDA BALLS    ; 2-6
4401: C9 06     05510           CMP #$06
4403: B0 09     05520           BCS ENDGAME
4405: 09 10     05530           ORA #$10     ; MAKE INTO CHR.
4407: A0 08     05540           LDY #$08
4409: 91 F0     05550           STA (WINDOW),Y
440B: 4C CA 42  05560           JMP GAME     ; GO BACK TO NEWBALL ROUTINE
                05570 ENDGAME
440E: A2 00     05580           LDX #$00
4410: AD 2A 42  05590           LDA BRICKS
4413: D0 01     05600           BNE .1
4415: E8        05610           INX          ; THEN 2ND MESSAGE
4416: 8A        05620 .1        TXA
4417: 0A        05630           ASL
```

```
4418: AA       05640          TAX
4419: BD C1 41 05650          LDA MSGTAB,X
441C: 85 F2    05660          STA MSG
441E: BD C2 41 05670          LDA MSGTAB+1,X
4421: 85 F3    05680          STA MSG+1
4423: A0 08    05690          LDY #$08
4425: B1 F2    05700 .2       LDA (MSG),Y
4427: 91 F4    05710          STA (LINE2),Y
4429: 88       05720          DEY
442A: 10 F9    05730          BPL .2
442C: AD 1F D0 05740 .3       LDA CONSOL
442F: C9 06    05750          CMP #$06
4431: D0 F9    05760          BNE .3
4433: 4C 30 42 05770          JMP START
```

CHAPTER 5

PLAYER—MISSILE GRAPHICS

Player-missile graphics differ from playfield graphics in two distinct ways. First, the players, which appear as semi-transparent overlays against the background, are memory independent of the playfield graphics. Thus, these sprites, as player-missiles are sometimes called, don't overwrite playfield memory and don't destroy the background graphics as they move around the screen. Second, since they are directly mapped to the screen by the CTIA/GTIA and the ANTIC hardware chips, positioning a sprite on the screen is extremely fast and accurate. They were designed with fast smooth motion in mind.

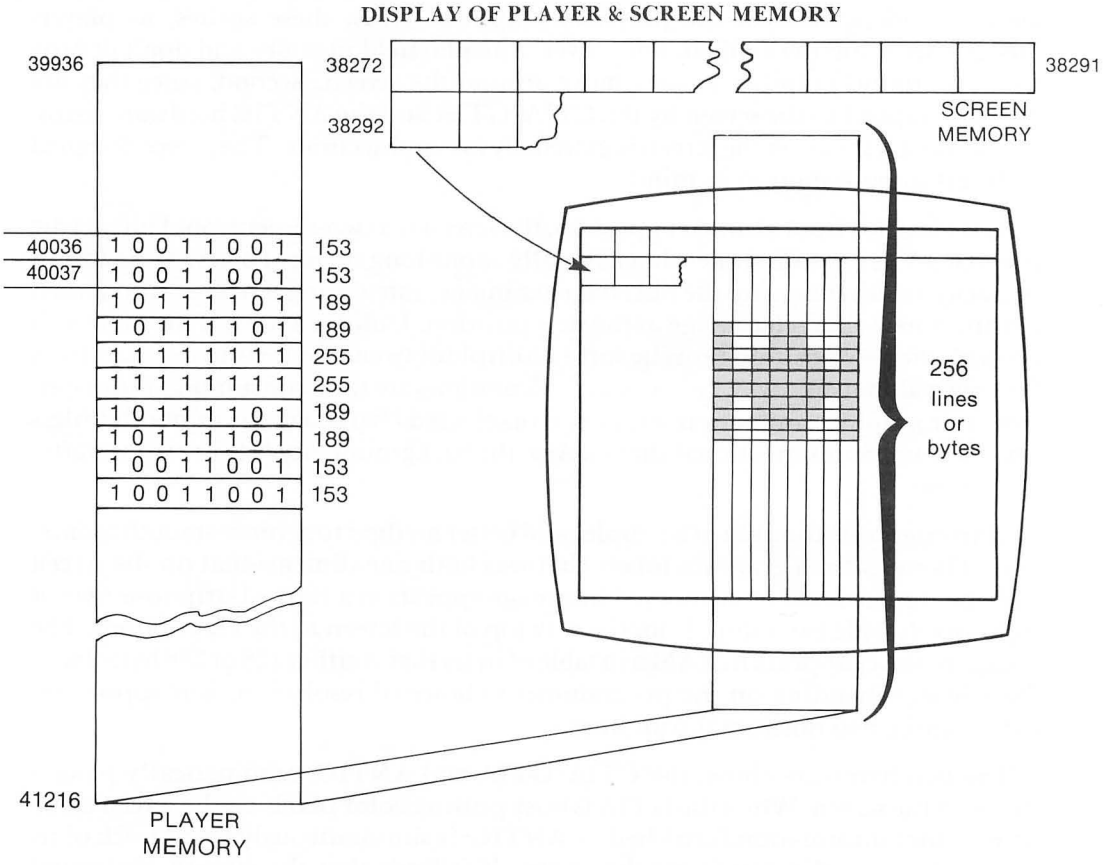
As we learned from character graphics, the screen is a two-dimensional image; the screen RAM is organized one-dimensionally as one long string of bytes. Animating a character requires erasing the old character image, calculating a new position, then writing a new character image at the new position. Unfortunately, if the motion is vertical, the new position must be some multiple of twenty or forty bytes apart from the original in memory. The necessary calculations are time consuming. Moreover, character graphics animation with 8 x 8 pixel sized characters isn't smooth unless you use numerous transitional shapes. Also the background needs to be restored after each move.

Atari engineers thought of a simpler and better method to achieve smooth animation. They created a graphics image that was both one-dimensional on the screen and one-dimensional in memory. This image appears as a vertical stripe one byte or eight pixels wide extending from the very top of the screen to the very bottom. The image or player appears in RAM as a table of bytes that is either 128 or 256 bytes long. Each byte, depending on the programmer's choice of resolution, is mapped into either one or two horizontal scan lines.

The two hardware chips, the CTIA/GTIA and ANTIC, automatically place a sprite on the screen. While the GTIA is busy putting color pixels on the screen based on graphics information furnished by ANTIC, it simultaneously keeps track of its current horizontal position on the screen. If it finds that the current horizontal position equals the value of the horizontal position register for the player, it asks ANTIC to give it a byte from the player-missile area of memory corresponding to the current scan line. It then interprets that byte as a series of on-off pixels or points, starting with the high bit on the left, and plots them in the selected player color. If the bit in the byte is on, it illuminates or plots a pixel, and if it is off it skips plotting the pixel. Areas of the player stripe that contains zeros or no player data are transparent to the background or playfield graphics. Scan lines that contain data form a solid image that overlays but does not affect the background image.

5 PLAYER MISSILE GRAPHICS

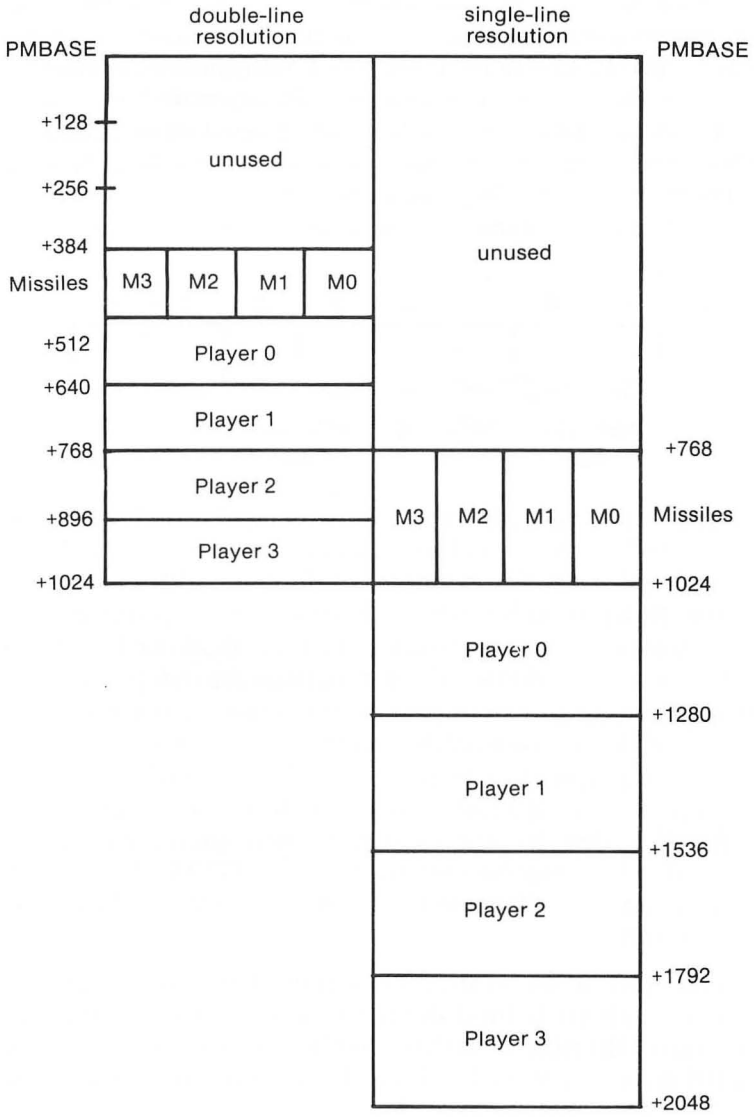
Player data is mapped in much the same way as character data is mapped in a character set. Where a character is limited to eight rows of data, a player stripe, depending on player resolution, consists of either 128 or 256 rows. Each byte corresponds to a row, and each of its eight-bit positions corresponds to the eight pixels that collectively form part of the player image. A bit that is on or has a one in it lights the corresponding pixel which maps from left to right, high bit to low bit. For example



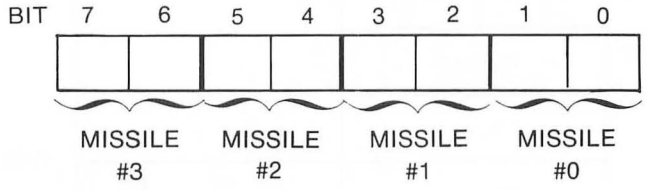
This data table, which is stored in the player-missile area of memory, is 256 bytes long. Actually, only 192 bytes corresponding to the screen's 192 scan lines are effectively used. A 256-byte area was chosen since it is one page of computer memory, and it is easier to find the start of a player's data when it begins on a page boundary.

The vertical starting position depends on the position in player memory. However, the 0th or first byte is mapped automatically by hardware to an area offscreen past where ANTIC is generating display list scan lines. Therefore, the first twenty or so bytes and the last thirty or so bytes are beyond the normal raster scan of a television set. A player that begins in the thirty-fourth position in player-missile memory will map to the screen starting at the second scan line. Players move vertically by moving the player data through the 256-byte page of player memory. The higher a player image is stored in memory, the lower it appears on the screen.

Atari's player-missile system consists of four players and four missiles that reside in a 2K block of memory known as the player-missile memory area. Think of



missiles as narrow players two pixels wide instead of eight pixels wide. Each of the four players has its own player-missile area of memory. Single-resolution players use 256 bytes or one page of memory, while double-resolution players use 128 bytes or half a page of memory. They are arranged sequentially so that player one follows player zero, etc. The four missiles, on the other hand, are stored in the same block of memory just below player zero. They are arranged in two-bit pairs with the 0th missile occupying the rightmost or lowest two bits, and the third missile the leftmost or highest two bits. While this arrangement is handy if you want to combine all four missiles to enable a fifth player, it presents a problem when moving a single missile vertically on the screen. If the missile data is moved in memory to correspond with a missile's screen movement, then portions of data for the other missiles that reside in those same bytes will also be moved. There is a Machine language solution to the problem that requires masking the bytes during movement, but this technique is unavailable from BASIC except by a complex USR function. Fortunately, horizontal movement is much easier. The designers incorporated a separate horizontal position register for each player and each missile. Even if all missiles are combined to enable a fifth player, each two-bit-wide missile band must be still set individually. Essentially, the fifth player is kept as a unit when it is moved, if each missile is positioned two units apart in the horizontal axis when it is moved.



The color and size of each player and its associated missile can also be specified. Size only affects the horizontal width of the player or missile. Widths can be normal, double, or quadruple size. For example, in double width, the GTIA chip begins double plotting the bytes on bits when it reaches the horizontal position register of the player or missile. Color is assigned to four shadowed player-missile color registers. Since these four additional color registers are independent of the playfield color registers, more colors can appear on the screen at any one time. While each player can have a different color, each missile is assigned the color of its corresponding player. Thus, if player three is green, missile three will also be green. The only exception to the rule is that if a fifth player is enabled, the four combined missiles use the color in playfield three's color register. There is rarely a conflict since playfield three is only used in four graphics modes. Since the GTIA plots both the player and playfield color pixels simultaneously, it can detect any overlap between graphics images on the screen.

Player priority can be set so that all or half of the players are in front of the playfield graphics, all are behind the playfield graphics, or some of the playfield colors are in front of the players with the rest behind. Any overlaps in position are of course returned in a series of read-only hardware locations called collision registers.

These are quite useful in game design. You can also enable bit 5 in the priority register (POKE 623,32) so that you obtain a third color when players 0 and 1 or 2 and 3 overlap. If you don't set the overlap option, the area of overlap will be black.

PLAYER—MISSILE REGISTERS

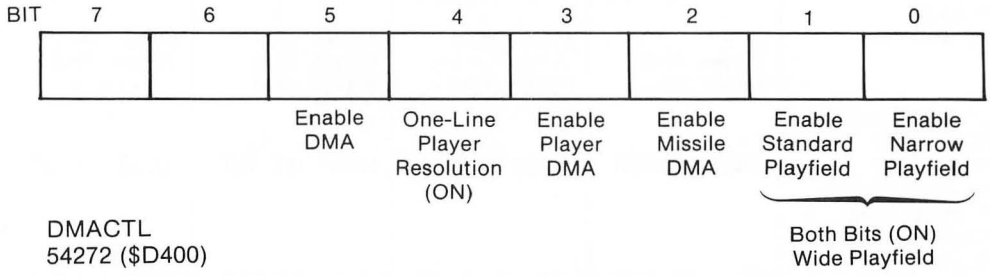
	PLAYER #0	PLAYER #1	PLAYER #2	PLAYER #3
PLAYER COLOR	704 ($2C0)	705 ($2C1)	707 ($2C3)	
PLAYER SIZE 0=normal 1=double 3=quadruple	53256 ($D008)	53257 ($D009)	53258 ($D00A)	53259 ($D00B)
HORIZONTAL POSITION	53248 ($D000)	53249 ($D001)	53250 ($D002)	53251 ($D003)
COLLISION (Read) PLAYER TO PLAYER	53260 ($D00C) ——— Player #1=2 Player #3=8	53261 ($D00D) Player #0=1 ——— Player #3=8	53262 ($D00E) Player #0=1 Player #1=2 Player #3=8	53263 ($D00F) Player #0=1 Player #1=2 ———
COLLISION (Read) PLAYER TO PLAYFIELD Playfield #0=1 Playfield #1=2 Playfield #2=4 Playfield #3=8	53252 ($D004)	53253 ($D005)	53254 ($D006)	53255 ($D007)
	MISSILE #0	MISSILE #1	MISSILE #2	MISSILE #3
MISSILE COLOR	Same as player	Same as player	Same as player	Same as player
MISSILE SIZE 0=normal 1=double 3=quadruple	53260 ($D00C) 0=normal 1=double 3=quadruple	8=normal 4=double 12=quadruple	32 = normal 16=double 48=quadruple	128 = normal 64=double 192=quadruple
HORIZONTAL POSITION	53252 ($D004)	53253 ($D005)	53254 ($D006)	53255 ($D007)
COLLISION (Read) MISSILE TO Player #0=1 Player #1=2 Player #2=4 Player #3=8	PLAYER 53256 ($D008)	53257 ($D009)	53258 ($D00A)	53259 ($D00B)
COLLISION (Read) MISSILE TO PLAYFIELD Playfield #0=1 Playfield #1=2 Playfield #2=4 Playfield #3=8	53248 ($D000)	53249 ($D001)	53250 ($D002)	53251 ($D003)

NOTE: Above values will set all missiles the same size.

NOTE: If missile sizes are set individually, then add the four different values. The combination is poked into ($D00C).

Enabling Player-Missile Graphics

There are numerous steps involved to enable player-missile graphics from both BASIC and Machine language. Beginners see it as a long list of mysterious POKEs, but there is a logical and explainable reason for each. First there are two electrical switches between Antic, GTIA, memory and the world, which tell them whether to do DMA (Direct Memory Access) automatically. The first, called DMACTL (Direct Memory Access Control), is shadowed at location 559 decimal ($22F). It enables ANTIC to fetch bytes automatically and plot them to the screen. When it is turned off with a zero, the screen is turned off because ANTIC can no longer fetch bytes from memory. It defaults normally to standard playfield graphics with double-line resolution for player-missiles turned off. This value is 34 decimal. The table below, which depends on which bit positions are set in DMACTL, summarizes the various modes.



DMA OFF	———	NO GRAPHICS	0
PM/OFF	———	NARROW PLAYFIELD	33
PM/OFF	———	STANDARD PLAYFIELD	34
PM/OFF	———	WIDE PLAYFIELD	35
PM/ON	DOUBLE-LINE RESOLUTION	STANDARD PLAYFIELD	46
PM/ON	DOUBLE-LINE RESOLUTION	WIDE PLAYFIELD	47
PM/ON	SINGLE-LINE RESOLUTION	STANDARD PLAYFIELD	62
PM/ON	SINGLE-LINE RESOLUTION	WIDE PLAYFIELD	63

The second switch GRACTL (Graphics Control) physically enables player-missile graphics. If we POKE a three into decimal location 54279 ($D01D), both players and missiles are turned on. In addition, we will need to set color, width, and horizontal positions for each player.

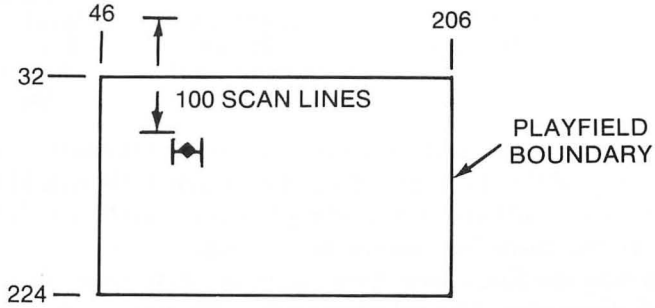
Next we need to reserve space for our 2K player-missile area in memory assuming we are using single-resolution P/M graphics. Since it needs to be on a 2K memory boundary, it is best to reserve space and put it above the screen and display list areas beginning at eight pages (2K) below the top of memory. This is accomplished by simply adjusting the top of memory pointer at location 106 decimal with a new value that is eight less than the original value. The computer now thinks top of memory is 2K lower than it actually is, and assigns screen memory and its accompanying display list below that when any graphics mode is set up.

The 2K player-missile area (single-line resolution) begins at the new top of memory which we just POKEd into location 106. Since ANTIC needs to know where we have put our P/M storage area, we need to POKE this value into location 54279 ($D407) known as PMBASE. This is the high byte value of the location. The low byte value is assumed to be zero since it is on a 2K page boundary.

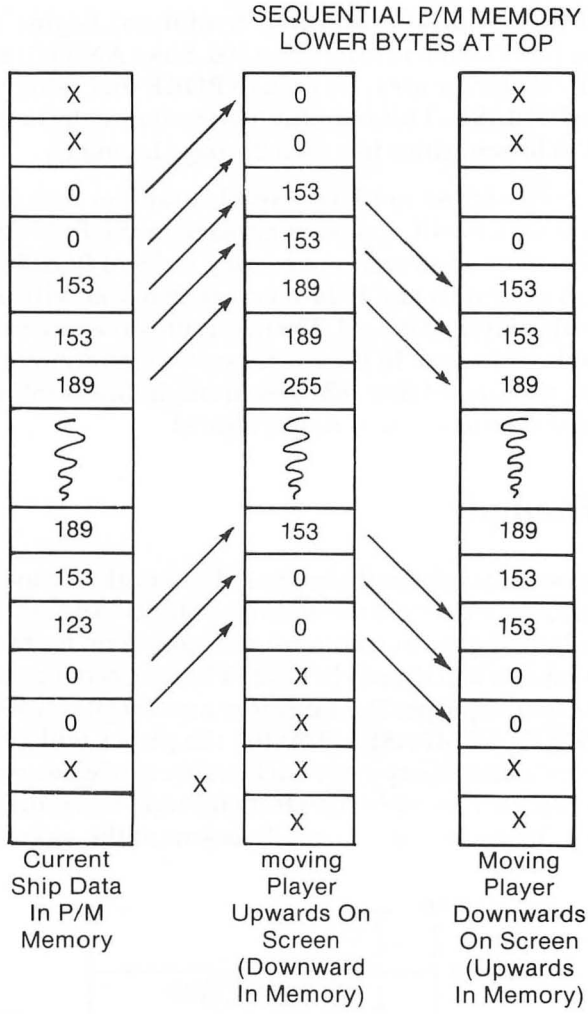
The memory area assigned to player-missile graphics may or may not contain miscellaneous data which will appear as garbage in the P/M stripes. The 2K area should be cleared to zeros. However, since a long series of POKEs in BASIC is rather slow, it is faster to clear only the P/M memory that you will actually use. In the example below, only player #0 is used. Therefore, only the area from PMBASE+1024 to PMBASE+1280 is set to zero. In the worst case where you were using the missiles and all the players, the unused first 768 bytes in single-line resolution or the first 384 bytes in double-line resolution need not be cleared.

Space Ship Example

Player #0 is a spaceship originally five scan lines high. To make it appear larger each scan line is plotted twice and the width is doubled. In addition, a pair of zeros has been placed at both the top and bottom of the player image for a total of fourteen bytes. The reason for this will shortly be clear. The fourteen player-missile data bytes are then POKEd into the proper P/M memory area for player #0. By POKEing the data starting at SPOT = PMBASE+1024+100 the player will begin 100 scan lines from the top. Since the first thirty-two scan lines are above the top line of text on the screen, our image begins sixty-eight lines from the top. Accounting for the two blank or zero lines of our image, our ship actually begins at the seventieth scan line.



For a player to move vertically, all of its data needs to be shifted in memory. Players can move downward on the screen by shifting the data upward in memory. Conversely a player moves upward on the screen by moving its data downward in memory. The latter case is definitely the easier direction because data can be moved from top to bottom without the danger of overwriting any of the bytes during the move. All fourteen bytes of data are shifted upward two scan lines from SPOT+I to SPOT+I-2. The two extra zero bytes on the end serve to erase the bottom two bytes of the ship's shape in its previous position. If this isn't done, the bottom sliver of the ship will remain as screen garbage after the move.



Moving the ship down the screen or upward in memory is slightly harder. If we start at the top of the shape as before, the first byte (0) would replace the third byte (153), and the second byte (0) would replace the fourth byte (153). Fine, but when we try to move the third byte to the fifth position, the original data (153) has been overwritten by the first move. Eventually all of the bytes will contain zeros and the shape will disappear. The solution to this dilemma is to start at the bottom of the shape and move each of the bytes upward in memory two places. Since you don't overwrite any shape information during the move, the shape remains intact. Again, the two extra zeros at the top serve to erase the tiny two-line sliver that would have remained after the move. It might seem strange that we deliberately added two zero bytes at the top and two at the bottom of our player when the rest of the storage area is obviously full of zeros. However, these could only be used if more bytes than that of our actual shape were moved, and only if we adjusted where the top or bottom of our shape begins. If we need to go to this trouble, it is easier to add extra leading and trailing zeros.

```
5 REM FIRST P/M EXAMPLE - MOVING A SHIP VERTICALLY UPWARDS FROM BASIC
10 POKE 106,PEEK(106)-8
20 PM=PEEK(106):PMBASE=PM*256
30 GRAPHICS 1+16
40 POKE 559,62:REM SET DMACTL - SINGLE LINE & STANDARD PLAYFIELD
50 POKE 53277,3:REM SET GRACTL - PLAYERS & MISSILES
60 POKE 54279,PM:REM TELL ANTIC PMBASE
70 POKE 53256,1:REM PLAYER #0 DOUBLE WIDTH
80 POKE 53248,100:REM PLAYER #0 HORIZONTAL POSITION
90 POKE 704,88:REM PLAYER #0 COLOR PINK
100 REM CLEAR OUT P/M AREA FOR PLAYER #0 ONLY
110 FOR I=PMBASE+1024 TO PMBASE+1280:POKE I,0:NEXT I
120 REM READ PLAYER #0 DATA INTO PLAYER AREA BUT STARTING 100 BYTES INTO PAGE
130 SPOT=PMBASE+1024+100
140 FOR I=0 TO 13:READ A:POKE SPOT+I,A:NEXT I
150 REM DELAY LOOP
160 FOR DE=1 TO 1000:NEXT DE
170 REM MOVE PLAYER DATA UP TWO SCAN LINES AT A TIME - TEN TIMES
180 FOR J=1 TO 10
190 FOR I=0 TO 13
200 POKE SPOT-2+I,PEEK(SPOT+I)
210 NEXT I
220 SPOT=SPOT-2:REM CURRENT PLAYER MEMORY LOCATION
230 NEXT J
235 REM MOVE PLAYER DATA DOWN TWO SCAN LINES AT A TIME - TWENTY TIMES
240 FOR J=1 TO 20
250 FOR I=13 TO 0 STEP -1
260 POKE SPOT+2+I,PEEK(SPOT+I)
270 NEXT I
280 SPOT=SPOT+2:REM CURRENT PLAYER MEMORY LOCATION
290 NEXT J
300 REM MOVE PLAYER HORIZONTALLY RIGHT
310 FOR I=100 TO 180
320 POKE 53248,I:NEXT I
330 GOTO 330
1000 REM PLAYER #0 DATA - SHIP
1005 DATA 0,0,153,153,189,189,255,255,189,189,153,153,0,0
```

A Player-Missile Machine Language Move Subroutine

It becomes apparent from this simple example that moving a player vertically using POKEs in BASIC is very slow. However, moving a player horizontally is fast because a horizontal position register was built into the hardware.

One clever approach to moving players fast vertically is to take advantage of BASIC's fast string-handling capabilities. These routines are just high speed Assembly language copy routines. The trick to using these routines is to fool the Atari into assigning the player-missile string data to the memory area assigned to player-missile graphics. Then player-missile graphics data can be moved simply with P$ = S$ type statements. While this technique allows the programmer to avoid Machine language subroutines, it is not the easiest method to learn or understand. Since the example and explanation would be intimidating to beginners, we will delay it until much later in this chapter.

Player-missile graphics can be speeded up tremendously if Machine language

subroutines are used. There have been numerous vertical blank player-missile subroutines published in the magazines. While most are effective in rapidly moving players vertically by one of several techniques, nearly all have neglected missile movement. This is unfortunate since moving one missile without disturbing the others requires using AND and ORA machine instructions on a bit level to mask off the other missiles during vertical movement. This is something that is beyond the capabilities of BASIC.

We have developed an easy-to-use Machine language subroutine that can handle both players and missiles simultaneously. The subroutine resides in page six of memory but is relocatable to virtually any free area of memory.

There has been much controversy on whether page six of memory is actually a safe area for placing user subroutines for BASIC programs. While it is true that inputting more than 128 characters into the text buffer will overwrite the beginning of page six, any program—especially a game that remains in control throughout—has no problem. In any case, the second half of page six $680-$6FF is always safe.

Our subroutine was designed for single-line resolution player-missile graphics. It requires the input of five variables: player number, player length, the old Y (vertical) position, the new Y position, and the new X (horizontal) position. The format is:

A = USR (1536, PLAYER #, PLAYER LENGTH, OLD Y, NEW Y, NEW X)

The number 1536 ($600) is the starting location of our subroutine. The players are numbered 0-3 and the missiles 4-7. Player #4 in our subroutine is actually missile #0, etc. The player length can only be as large as sixty-four lines. This limitation will become clear when we explain how the subroutine works. The values for both the X and Y positions can be anything from 0-255. However, you should be aware that only X values in the range of 48-207 and Y values in the range 32-223 are within the bounds of the 40 by 24 text screen or playfield graphics area. The range is actually slightly larger, but visibility depends on the overscan of your television set. While the OLDY value is the programmer's choice upon initialization, it must be set equal to the NEWY position after each USR call. If OLDY isn't set to NEWY immediately after the subroutine call, unpredictable graphics will begin to appear in the player stripe when the subroutine is used again. The reason is that the subroutine zeros out the shape at its previous position OLDY before drawing the shape again at its new position NEWY. If OLDY isn't set to the player's last vertical position, the subroutine will miss zeroing the shape at the old position. Most likely, you will get two shapes or, worse yet, a sliver left at the previous position.

A Player-Missile Example

The next example is a simple demonstration of the speed, versatility and ease of use of our player-missile subroutine. This two-part example is instructive in several regards. First, it shows how a simple spaceship capable of firing missiles can be joystick controlled. Even its simple in-line code, which allows either the ship to move or the missile to move, but not both simultaneously, gives some insight into

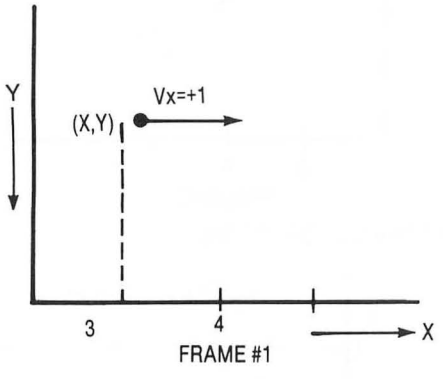
the difficulty in designing even the simplest games. Second, the concept of priority, or which player or playfield is to be drawn in front of another is demonstrated. But first I think a discussion of how objects move might be helpful, especially to beginners.

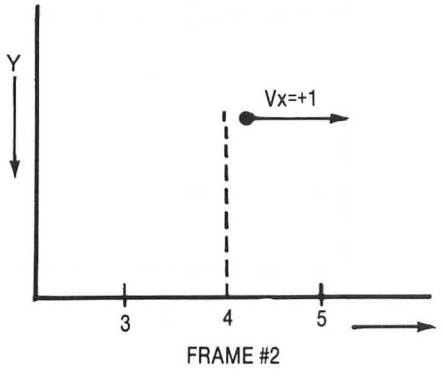
Dynamics of Objects in Motion

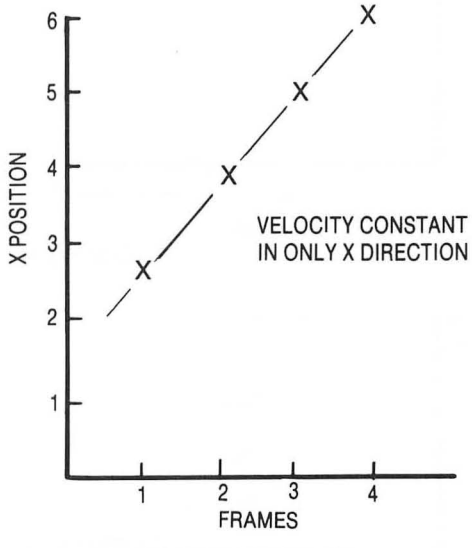
Any object in motion, whether it is simulated on a video screen or moves in the real world, is subject to the laws of physics. Readers will immediately cringe at the thought of advanced mathematics, mainly calculus. But calculus is merely a method of calculation that involves the summation of many small bits and pieces of a body's velocity and acceleration to determine the actual distance an object travels. Fortunately, the computer automatically divides our time frame into analogously small units, or animation frames.

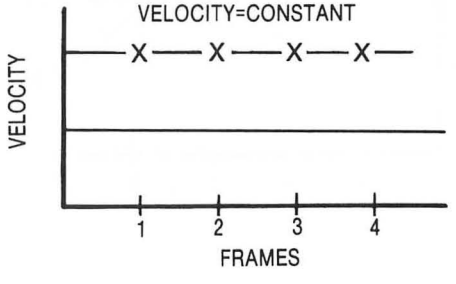
Let's examine an object in simple linear motion. The object is initially at rest. It is then given a horizontal velocity of one unit to the right. Thus the velocity is +1 unit/time frame. During each animation frame, the object moves +1 units to the right.

An object's direction of travel and its magnitude is represented by a line segment called a vector. An object's velocity always points in the direction of travel. Our object shown below has a velocity of +1 units/time frame, so that the velocity is pointing to the right. Since the velocity vector is to the right, the object moves to the right +1 unit/frame.









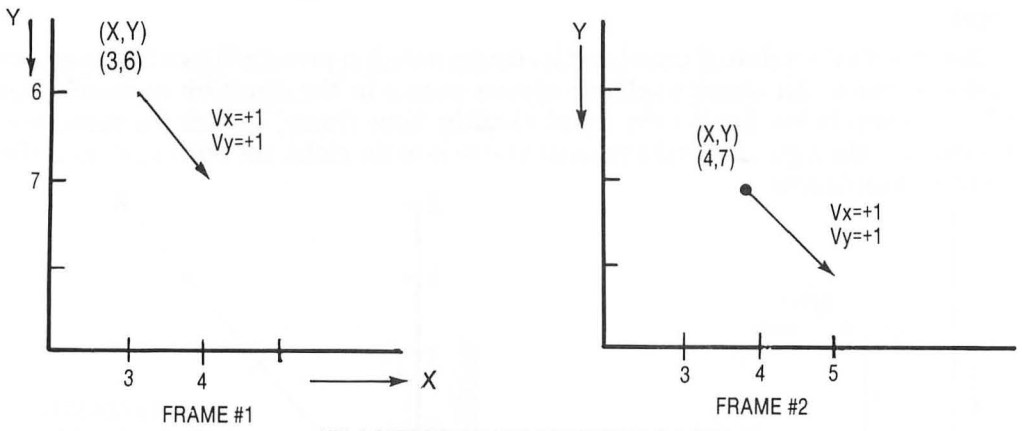
Similarly, an object that moves diagonally downward and to the right has a positive velocity vector in both the X and Y directions. This velocity vector is a combination of the velocity components in both the X and Y directions. The object will continue moving in the direction of the velocity vector until either VX or VY changes. For example, if VX becomes zero, the object will begin to move straight down in the direction of the new velocity vector.

This can be formalized into equations for each of the two screen directions X and Y.

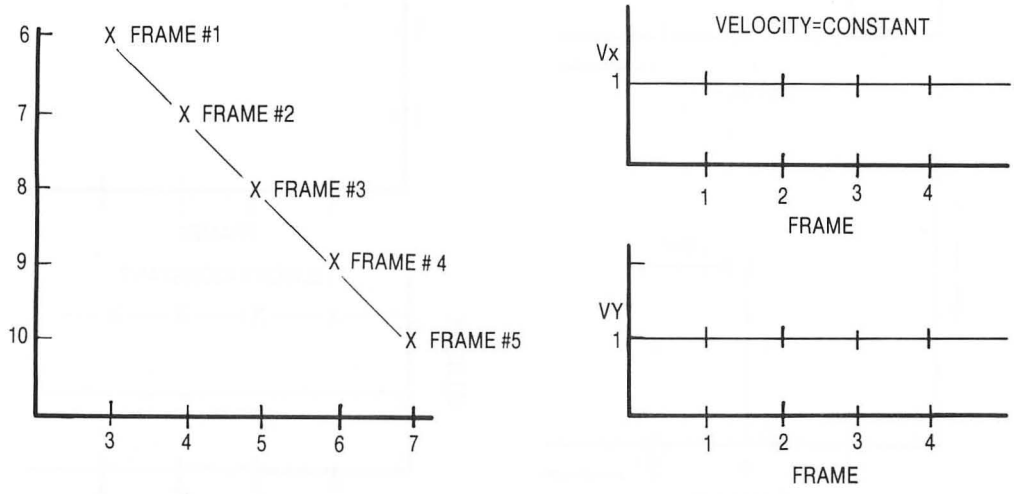
VX = +1	Velocity is constant in X direction.
X = X + VX	New position is the old position plus the change in position (velocity).

Likewise:

VY = +1	Velocity is constant in Y direction.
Y = Y + VY	New position is the old position plus the change in position (velocity).



VELOCITY CONSTANT IN BOTH X & Y DIRECTION

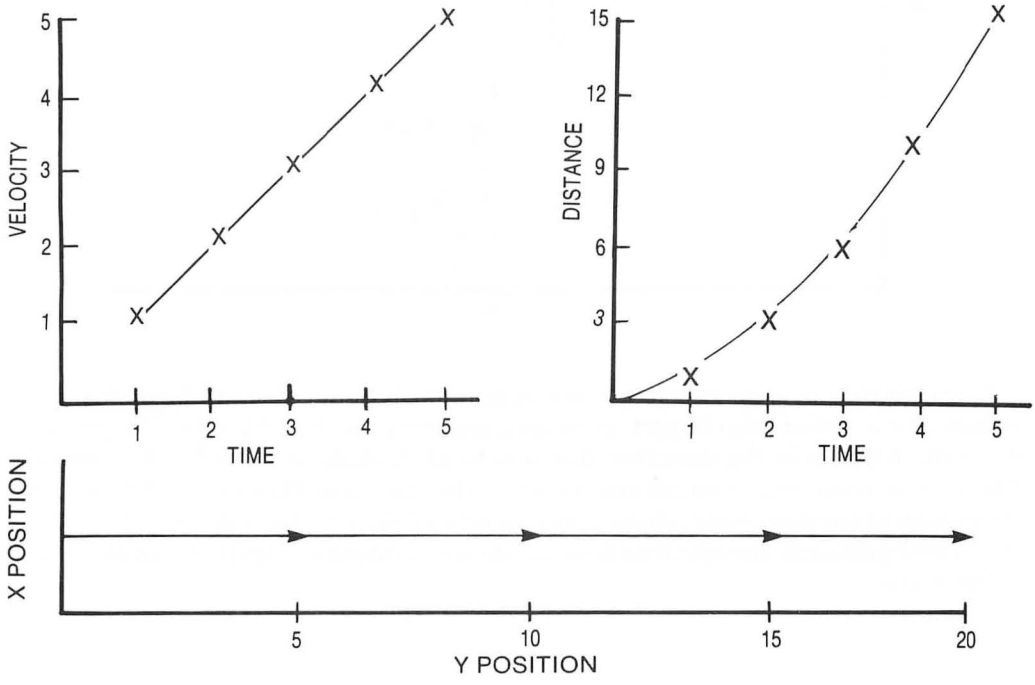


While this simplistic method of moving objects by instantaneous changes in direction and velocity works on the video screen, objects in the real world don't behave this way. Take the family car for example. You step on the gas pedal, and the car's velocity begins to climb steadily. The distance traveled each second begins to increase as the velocity increases. When the car is only going 15 MPH the car travels only 22 feet each second, but when the car reaches 60 MPH it travels 88 feet each second.

The driving force that speeds up our car is called acceleration (V = V + A). Acceleration can be constant as in a rocket's thrust, or like gravity which pulls a falling object to Earth. For a screen object's motion to appear realistic, especially in the case of a falling bomb, acceleration must be taken into account. When a bomb drops, its vertical velocity increases with time. If there were no wind resistance, the bomb's vertical velocity would increase until its impact on the target.

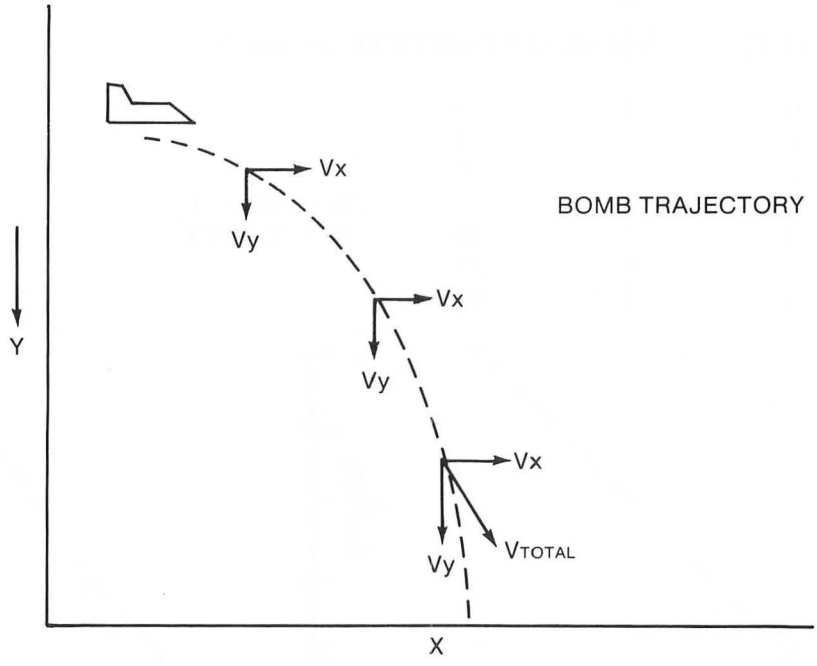
If a constant force was suddenly applied to a stationary object such that it accelerated downward with an increase in velocity of one unit/ frame, the distances moved would grow substantially.

TIME	VELOCITY	POSITION (distance)	
0	0	0	
1	1	1	
2	2	3	
3	3	6	VX = VX + 1
4	4	10	X = X + VX
5	5	15	
6	6	21	



The plot of the trajectory of a falling bomb is shown below. The trajectory, neglecting wind resistance, forms a curve that is called "parabolic." There are two components to the velocity vector. VX in the X direction is a constant set equal to the plane's forward velocity. VY in the Y direction grows larger with time as gravity accelerates it in the downward direction. This same effect can be observed by dropping a ball from the second or third story of a building. At first, the ball falls slowly, but then it begins falling faster. Observers at ground level will note the acceleration of the moving ball just before it bounces. The summation of the two velocity vectors determines the resultant direction of an object's motion for each animation frame. Since the VY vector grows larger with each frame, the total velocity vector begins to point downward. Eventually, the bomb will be falling almost straight down. Thus:

```
VX = CONST              X = X + VX
VY = VY + GRAVITY       Y = Y + VY
```



In the above cases, the acceleration was non-existent or constant. However, as we will see at the end of this chapter, even simple games involving a steerable spaceship that can be thrust in the direction that it is headed, must use variable acceleration. The rate or value may remain constant but the direction changes. While only the beginning of the discussion above is necessary to follow the second example, I hope it will give you some insight into how an object's motion is simulated on the Atari's video screen.

Program Initialization

Our first example, a joystick-controlled single ship capable of firing missiles in eight directions, takes advantage of our player-missile Machine language subroutine. As in the first example in this chapter, a 2K block of memory needs to be reserved for player-missile graphics. The top of memory pointers are moved down eight pages, and the player-missile base PMBASE is set equal to the top of memory. BASIC then sets up the screen area and its display list just below. Graphics 1 was chosen because it uses little screen memory and because we will need to write some large characters to the screen for the second part of our example.

Next, our two Machine language subroutines need to be POKEd or placed into memory. The first subroutine is the player-missile subroutine. It is 151 bytes long. Since it begins at the start of page six ($600) or 1536 decimal, a simple FOR...NEXT loop that reads the data statements and POKEs memory, will do the job. The second Machine language subroutine is a clear memory routine. It will automatically clear player-missile memory to zero in a fraction of a second. It replaces a slow thirty second long FOR...NEXT loop which would POKE a zero into each sequential memory location from PMBASE+0 to PMBASE+2047. It also resides in page six of memory, beginning at $6A0 or 1696 decimal. It is twenty-six bytes long.

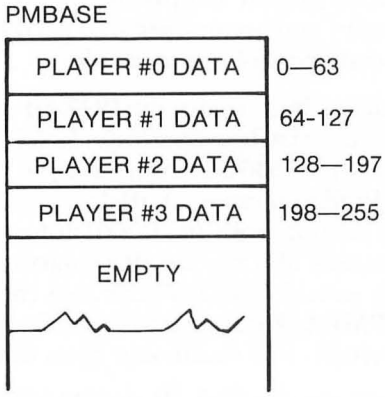
We must activate player-missile graphics next. We will set DMACTL for single-line resolution and GRACTL for both players and missiles. Player #0 is set to double width by writing a zero to location 53256, and player #2, which will be used in the second part of this example, is set to double width by writing a one to location 53258. The missiles are set to regular width, and the color of each of the players is selected. The priority GPRIOR shadowed at location 623 decimal is set so that player #0 is in front of the playfield graphics and player #2 is behind. We will talk more about this location during the discussion of the second half of this example. Last, we tell ANTIC our player-missile address by writing it to PMBASE at decimal location 54279. Since our subroutine also needs the location of PMBASE, its high byte is POKEd into location 1686 which is the last memory location of our subroutine. We could have passed it to the subroutine via the USR function, but then you would have had to type the value in each time you used the subroutine.

Player Shapes Using the P/M Subroutine

We then store the two player shapes in our player shape storage area. You will recognize the first shape from the first example. It is our spaceship. The second shape is a solid block ten bytes high. Normally, you would want to POKE your shapes into the proper place in the player-missile memory area. However, when we designed the subroutine, we felt that the best and fastest approach would be to erase the shape at its old location before redrawing it at the new location. The traditional approach was to do a memory move of the entire shape including leading and trailing zero bytes, to insure pieces aren't left behind. If a shape were moved vertically more than one or two scan lines, a series of small memory moves had to be done sequentially.

5 PLAYER MISSILE GRAPHICS

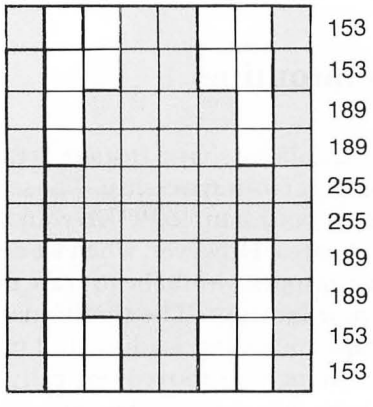
In order to use our faster subroutine, we had to find a safe place to store our shapes in memory, no matter where you put our subroutine, the player-missile area, and screen memory. Fortunately, the first 768 bytes of the player-missile memory area are unused. We decided to use the first page or 256 bytes for the player storage area. This limits the size of each of the four players to sixty-four bytes. Since sixty-four bytes or scan lines make up nearly one-third of the screen, this limitation really doesn't cause any problems.



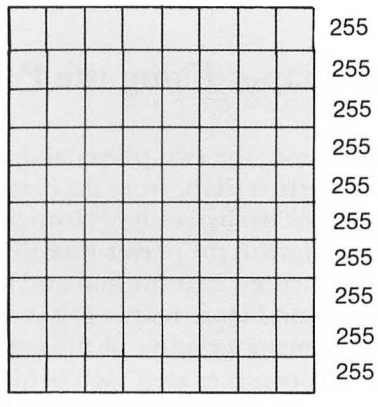
PLAYER STORAGE AREA
SINGLE RESOLUTION

Before any shapes are stored in this "safe" area, player-missile memory should be cleared or zeroed with the Machine language subroutine stored in page six of memory at 1536 decimal. The only parameter passed via the USR function is the starting memory address PMBASE. This routine was specifically designed to clear eight pages of memory, precisely that of a 2K block of player-missile memory. If you neglect to clear it, random patterns will most likely surround your shapes in the player missile stripes.

The first shape, our spaceship, which is player #0, is POKEd into the first ten bytes of the player-missile storage area from PMBASE+0 to PMBASE+9. The second shape, the solid block, which is player #2, is stored beginning at PMBASE+128.



PLAYER#0



PLAYER #2

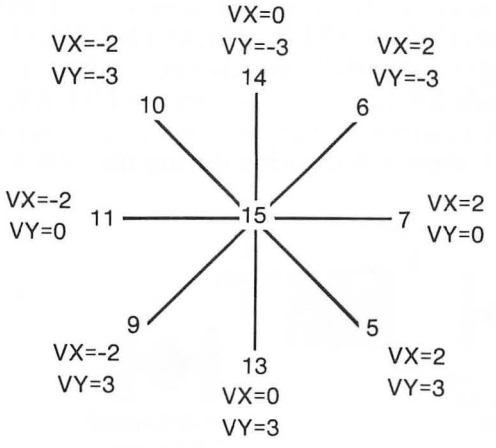
The spaceship (player #0) must have an initial starting position and starting velocity. While it is obvious that the present position X0=100, Y0=80 is important, its previous Y axis position must have an initial value despite the fact that there couldn't have been a previous one. Setting Y0OLD=80, the value of our present Y position, will suffice. Likewise, the same is true for player #0's missile. YM0OLD=10, a value that insures that it is off screen.

Joystick Controlled Ship Movement

The spaceship's velocity is joystick-controlled. Pushing the stick in any direction instantaneously changes the ship's velocity. If the stick is pushed to the right, the ship is given a horizontal velocity of 2 units/frame. Since there is no vertical component to the ship's velocity, the ship will move to the right 2 units/frame. It will continue in that direction until either it collides with the screen's boundary, or a new joystick input gives the ship a new velocity vector. The equations that control the ship's X and Y position are as follows.

$$X0 = X0 + VX0$$
$$Y0 = Y0 + VY0$$

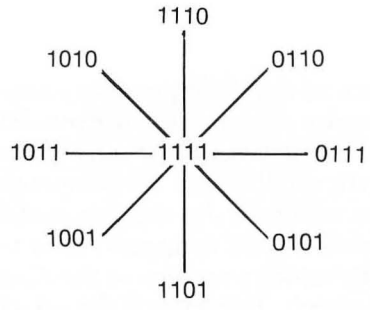
There is a separate pair of velocity vectors for each of the eight possible joystick directions. The absolute values on each of the axis are not the same because pixel size is rectangular rather than square. Objects with equal velocities along both axis tend to move faster along the horizontal axis than along the vertical axis. To compensate for this, we set VX = 2 and VY = 3. Diagonal velocities, which are the vector sum of the two velocity components, are faster than anticipated. While it appears that you could correct this by using fractional values for both velocity vectors on the diagonals, you can't move part of a pixel position in either axis. However, if the velocity vectors were much larger you could correct the diagonal velocity by using smaller whole numbers. The diagram below shows the velocity component values for each of the joystick directions.



Joystick Direction & Velocity Vectors

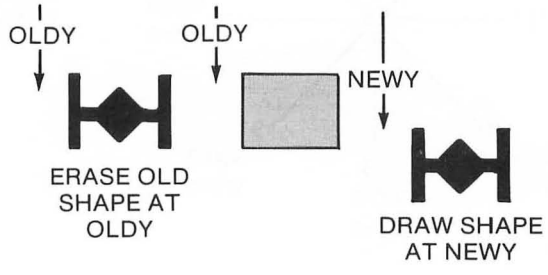
The only way to set the ship's two velocity vectors for each joystick direction is to test the value of Z0 = STICK(0) against each of its possible values. When the stick is centered (Z0=15) the velocity remains the same and we skip the remainder of the tests. The other eight possiblities are tested in a series of IF statements. If a match is found for any direction, new velocity vectors are substituted. While it appears to be a cumbersome method, there is no better method.

I'm sure many readers wonder why Atari BASIC returns such a strange and illogical set of values for joystick directions. STICK(0) through STICK(3) reflect values returned in the PIA chips locations 54016 and 54017 ($D300,$D301). When the joysticks are centered, each of the four bit locations for each joystick are on (set to one). When the joystick is pushed in any direction, its appropriate bit position is turned off (set to zero). For example, if the joystick is pushed to the right, the fourth bit from the right is turned off. The remaining three on bits add up to seven. Diagonal joystick movements have some combination of two bits turned off. The diagram below shows all of the possible bit patterns.



	8 Right	4 Left	2 Down	1 Up	
CENTERED	1	1	1	1	15 $0F
UP	1	1	1	0	14 $0E
DOWN	1	1	0	1	13 $0D
LEFT	1	0	1	1	11 $0B
RIGHT	0	1	1	1	7 $07

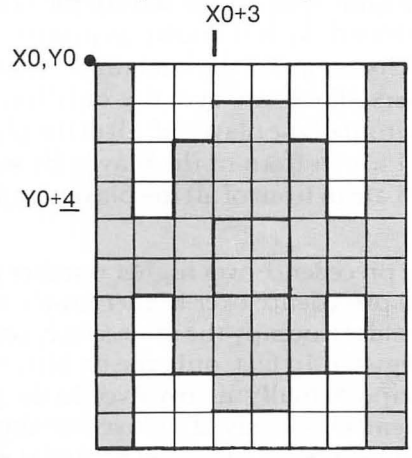
After the joystick is read, the player's position is updated and checked so that it does not exceed the screen boundaries in either direction. A USR call to our player-missile Machine language subroutine will move the player shape into the proper place in memory so that it appears at the choosen X,Y screen coordinates. The format as discussed earlier is A = USR (1536, PLAYER #, PLAYER LENGTH, OLD Y, NEW Y, NEWX). The player length is 10, OLDY equals Y0OLD in this example, NEWY equals Y0, and NEWX equals X0. Immediately after the USR call, update the OLDY position with the NEWY position so that you don't have to worry about losing this value when you calculate your new position during the next frame.



Missile Movement

Pressing the joystick button fires a missile. STRIG (0) is normally set to a one value when the button isn't pressed. But when the button is pressed it becomes zero and the program branches to a second joystick read routine at line 185. It is nearly identical to the first, with the exception that the velocity vectors are nearly double in value; VXM0=5 and VYM0=6. This ratio produces nearly identical apparent speeds in both the horizontal and vertical axis. Since the missile needs to be given a direction to fire, it branches past the missile routine if the joystick is centered.

The missile's initial starting position is at the center of our ship. Since X0, Y0 are the coordinates of the ship's missile at the center of the ship, the missile's initial position equals the current ship's position plus the correction. Thus, XM0=X0+3 and YM0=Y0+4. The missile's position is updated each frame the same way the



ship's position is updated each frame. The new position equals the old position plus the missile's velocity vector.

$$XM0 = XM0 + VXM0$$
$$YM0 = YM0 + VYM0$$

Unlike the ship, however, the missile's velocity vector can't be changed once the missile is fired. Missile movement is in a closed loop that exits only when the missile reaches the boundaries of the screen. This closed loop not only prevents you from firing a second missile before the first has reached the end of its travel, it also prevents the ship from moving while the missile is in motion. This is a good example of how simplistic code that branches to either one event or the other creates an unrealistic effect. The solution, which will be explained in the next example, requires the ship's movement code to be placed in line with the missile's movement code, and conversely the missile's movement code to be placed in line with the player's movement code. This apparent overkill, in fact, allows the ship to be steered once the missile is launched.

The missile is plotted via the USR call to our player-missile subroutine. Missile #0, which is two pixels wide, is player #4 and its height is set to two scan lines to make

it square shaped. Its OLDY position is YM0OLD, and its new position NEWX, NEWY is XM0, YM0. Again, once the subroutine is called via the USR call, it is important to immediately set the missile's old position equal to its present position.

Finally, if the missile does reach the screen boundaries, it is necessary to place it out of view beyond the scan of the television set. Since the missiles can't be simply shut off, you should set the horizontal register to either a small number or a large number. Since XM0=10 is offscreen, it will suffice.

Priority Demonstration

The second part of the example is a demonstration of player versus playfield priority. Whenever two independent objects such as two players or a player and a playfield object appear at the same spot on the screen, the GTIA must decide who gets displayed first and who second. This is known as priority. There is a register called GPRIOR shadowed at location 623 decimal ($26F) that selects which screen object is in fromt of the others. There are actually only four possible selections. Either all of the players are in front of the playfield, all of the playfields are in front of the players, players #0 and #1 are in front of the playfields with players #2 and #3 behind, or playfields #0 and #1 are in front of all the players with playfields #2 and #3 behind.

Lower number players take precedence over higher number players and, likewise, lower number playfields take precedence over higher number playfields. Actually, there is no way for two playfields to occupy the same space, since each color pixel is assigned to a specific color register. In fact, only the on bits, or those portions that show in the player-missile stripe, actually are involved in the priority conflict. The off bits are ignored by the system. Obviously, if a player has the shape of a donut, the background or another higher numbered player set behind it will show through the hole.

ORDER OF INCREASING PRIORITY

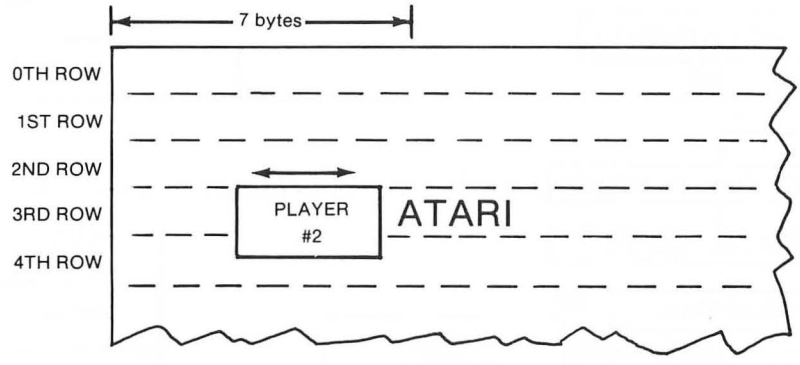
HIGHEST ← → LOWEST

POKE 623,1 BIT 0 SET	PLAYER #0	PLAYER #1	PLAYER #2	PLAYER #3	PLAYFIELD #0	PLAYFIELD #1	PLAYFIELD #2	PLAYFIELD #3
POKE 623,2 BIT 1 SET	PLAYER #0	PLAYER #1	PLAYFIELD #0	PLAYFIELD #1	PLAYFIELD #2	PLAYFIELD #3	PLAYER #2	PLAYER #3
POKE 623,4 BIT 2 SET	PLAYFIELD #0	PLAYFIELD #1	PLAYFIELD #2	PLAYFIELD #3	PLAYER #0	PLAYER #1	PLAYER #2	PLAYER #3
POKE 623,8 BIT 3 SET	PLAYFIELD #0	PLAYFIELD #1	PLAYER #0	PLAYER #1	PLAYER #2	PLAYER #3	PLAYFIELD #2	PLAYFIELD #3

Unfortunately, player #0 always has priority over the other three players. There is no method to change the priority between the individual players other than to switch the player data between player-associated areas of memory.

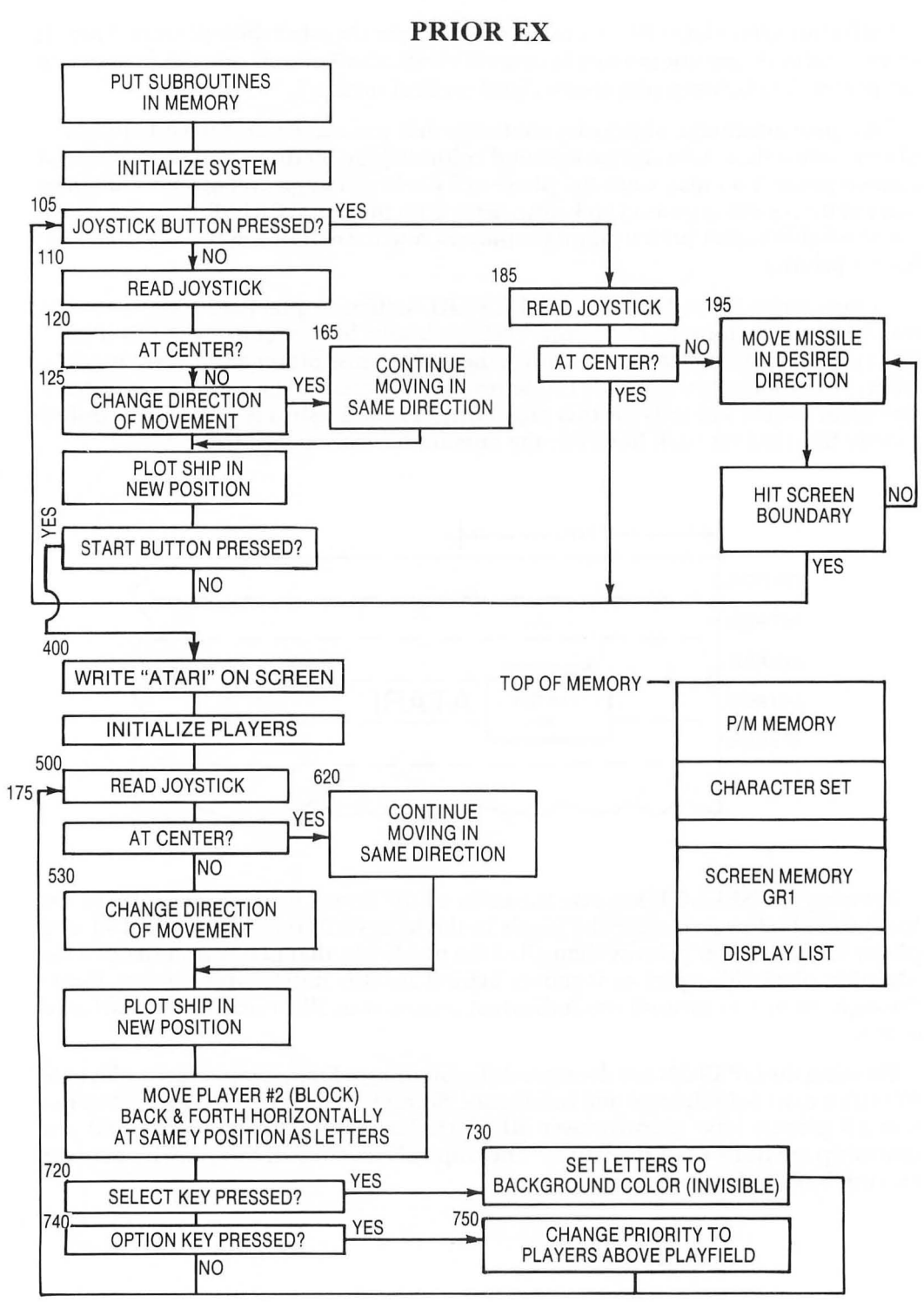
The great advantage of priority control is that you can control what happens to players when they meet the background color display. Perhaps you have an aerial combat game. You may want the plane to fly behind clouds, yet always remain in front of the scrolling ground far below. Setting bit three in GPRIOR will give two of the playfields higher priority than the players, and the remaining two playfields the lowest priority.

In our example, we have the word ATARI written in playfield #0, a spaceship (player #0), and a moving block (player #2). Initially, bit 1 is set in GPRIOR (POKE 623,2) so that player #0 moves in front of the letters while player #2 moves behind the letters. This part of the example can be reached by pressing the START key. The IF statement in line 175 tests for this possibility. The spaceship is joystick-controlled exactly like the first part; however, the missiles have been disabled.



Pressing the SELECT key sets the color of the letters to the same color as the background. However, since the pixels in the letters still refer to playfield #0, and player #2 had a lower priority than all of the playfields, that player is masked by the invisible playfield object as it moves behind it. Our much larger player shows through the spaces around the individual letters, thus illuminating the darkened object.

Pressing the OPTION key does two different things. First, it restores the playfield #0's color to its default color and luminance. Second, it changes the GPRIOR to 1 so that all players have priority over all playfields. Now both our block and our spaceship are in front of the letters. The ship (player #0) still has priority over the moving block (player #2).


PRIOR EX
PUT SUBROUTINES IN MEMORY
INITIALIZE SYSTEM
105
JOYSTICK BUTTON PRESSED?
YES
110
NO
READ JOYSTICK
120
AT CENTER?
165
CONTINUE MOVING IN SAME DIRECTION
125
NO
YES
CHANGE DIRECTION OF MOVEMENT
PLOT SHIP IN NEW POSITION
YES
START BUTTON PRESSED?
NO
185
READ JOYSTICK
AT CENTER?
NO
YES
195
MOVE MISSILE IN DESIRED DIRECTION
HIT SCREEN BOUNDARY
NO
YES
400
WRITE "ATARI" ON SCREEN
INITIALIZE PLAYERS
500
175
READ JOYSTICK
620
AT CENTER?
YES
CONTINUE MOVING IN SAME DIRECTION
NO
530
CHANGE DIRECTION OF MOVEMENT
PLOT SHIP IN NEW POSITION
MOVE PLAYER #2 (BLOCK) BACK & FORTH HORIZONTALLY AT SAME Y POSITION AS LETTERS
720
SELECT KEY PRESSED?
YES
730
SET LETTERS TO BACKGROUND COLOR (INVISIBLE)
740
OPTION KEY PRESSED?
YES
750
CHANGE PRIORITY TO PLAYERS ABOVE PLAYFIELD
NO
TOP OF MEMORY
P/M MEMORY
CHARACTER SET
SCREEN MEMORY GR1
DISPLAY LIST

```
5 REM DOUBLE EXAMPLE - ONE SHIP FIRING MISSILES & PRIORITY DEMONSTRATION
10 POKE 106,PEEK(106)-8
20 GOTO 1000
49 REM INITILIZE STARTING POSITIONS
50 X0=100:Y0=80
57 VX0=0:VY0=0
99 REM PLAYER #0
105 IF STRIG(0)=0 THEN 185
110 Z0=STICK(0)
120 IF Z0=15 THEN 165
124 REM UPDATE VELOCITY VECTORS
125 IF Z0=14 THEN VX0=0:VY0=-3
130 IF Z0=13 THEN VX0=0:VY0=3
135 IF Z0=10 THEN VX0=-2:VY0=-3
140 IF Z0=9 THEN VX0=-2:VY0=3
145 IF Z0=11 THEN VX0=-2:VY0=0
150 IF Z0=7 THEN VX0=2:VY0=0
155 IF Z0=6 THEN VX0=2:VY0=-3
160 IF Z0=5 THEN VX0=2:VY0=3
164 REM UPDATE POSITION PLAYER0
165 X0=X0+VX0:Y0=Y0+VY0
166 IF X0<46 THEN X0=46
167 IF X0>198 THEN X0=198
168 IF Y0<24 THEN Y0=24
169 IF Y0>216 THEN Y0=216
172 A=USR(1536,0,10,Y0OLD,Y0,X0)
173 Y0OLD=Y0
175 IF PEEK(53279)=6 THEN 400:REM IF START PRESSED GOTO 2ND PART DEMO
180 GOTO 105
184 REM FIRE MISSILE
185 Z0=STICK(0)
190 IF Z0=15 THEN 270
195 IF Z0=14 THEN VXM0=0:VYM0=-6
200 IF Z0=13 THEN VXM0=0:VYM0=6
205 IF Z0=10 THEN VXM0=-5:VYM0=-6
210 IF Z0=9 THEN VXM0=-5:VYM0=6
215 IF Z0=11 THEN VXM0=-5:VYM0=0
220 IF Z0=7 THEN VXM0=5:VYM0=0
225 IF Z0=6 THEN VXM0=5:VYM0=-6
230 IF Z0=5 THEN VXM0=5:VYM0=6
237 XM0=X0+3:YM0=Y0+4:REM CORRECTS MISSILE START TO CENTER OF SHIP
239 REM UPDATE POSITION MISSILE 0
240 XM0=XM0+VXM0:YM0=YM0+VYM0
245 IF XM0<40 THEN 260
246 IF XM0>207 THEN 260
247 IF YM0<20 THEN 260
248 IF YM0>220 THEN 260
250 A=USR(1536,4,2,YM0OLD,YM0,XM0)
255 YM0OLD=YM0:GOTO 240
259 REM REMOVE MISSILE TO OFF SCREEN
260 A=USR(1536,4,2,YM0OLD,YM0OLD,10)
270 GOTO 105
398 REM 2ND PART OF DEMO - PRIORITY
399 REM WRITE "ATARI" ON SCREEN
400 SCREEN=PEEK(88)+PEEK(89)*256
410 OFFSET=20*3+7
412 POKE SCREEN+OFFSET,33:POKE SCREEN+OFFSET+1,52:POKE SCREEN+OFFSET+2,33
415 POKE SCREEN+OFFSET+3,50:POKE SCREEN+OFFSET+4,41
420 REM INITILIZE PLAYERS
425 X0=100:Y0=150:VX0=0:VY0=0
430 X2=70:Y2=56:VX2=4
```

```
435 X2OLD=80
500 Z0=STICK(0)
510 IF Z0=15 THEN 620
520 REM UPDATE VELOCITY VECTORS
530 IF Z0=14 THEN VX0=0:VY0=-3
540 IF Z0=13 THEN VX0=0:VY0=3
550 IF Z0=10 THEN VX0=-2:VY0=-3
560 IF Z0=9 THEN VX0=-2:VY0=3
570 IF Z0=11 THEN VX0=-2:VY0=0
580 IF Z0=7 THEN VX0=2:VY1=0
590 IF Z0=6 THEN VX0=2:VY0=-3
600 IF Z0=5 THEN VX0=2:VY0=3
610 REM UPDATE POSITION PLAYER0
620 X0=X0+VX0:Y0=Y0+VY0
625 IF X0<46 THEN X0=46
630 IF X0>198 THEN X0=198
635 IF Y0<24 THEN Y0=24
640 IF Y0>216 THEN Y0=216
650 A=USR(1536,0,10,Y0OLD,Y0,X0)
660 Y0OLD=Y0
670 REM MOVE BLOCK (PLAYER#2)
680 X2=X2+VX2
690 IF X2>180 THEN VX2=-4
700 IF X2<70 THEN VX2=4
710 A=USR(1536,2,10,Y2OLD,Y2,X2)
719 REM TEST IF SELECT KEY PRESSED & IF SO SET LETTERS TO BACKGROUND COLOR
720 IF PEEK(53279)<>5 THEN 740
730 POKE 708,PEEK(712)
739 REM TEST IF OPTION KEY PRESSED & IF SO CHANGE PRIORITY
740 IF PEEK(53279)<>3 THEN 500
750 POKE 623,1
760 POKE 708,40:REM DEFAULT COLOR
770 GOTO 500
1000 REM SETUP
1005 PM=PEEK(106):PMBASE=PM*256
1010 GRAPHICS 1+16
1020 SETCOLOR 2,0,0:REM SET BACKGROUND BLACK
1052 REM POKE IN PM ROUTINE
1053 FOR I=0 TO 150:READ X:POKE 1536+I,X:NEXT I
1055 REM POKE IN PM CLEAR ROUTINE
1056 FOR I=0 TO 25:READ X:POKE 1696+I,X:NEXT I
1270 REM ACTIVATE P/M
1300 POKE 559,62:REM SET DMACTL -SINGLE LINE
1302 POKE 53277,3:REM SET GRACTL -PLAYERS&MISSILES
1320 POKE 53256,0:POKE 53258,1:REM PLAYER 0 REGULAR WIDTH; PLAYER 2 DOUBLE WIDTH
1325 POKE 53260,0:REM MISSILES REGULAR WIDTH
1330 POKE 704,152:REM PLAYER0 BLUE GREEN LUM 8
1332 POKE 706,52:REM PLAYER2 RED ORANGE LUM 4
1334 POKE 623,2:REM PRIORITY SELECTED HAS PLAYER 0,1 PLAYFIELD THEN PLAYER 2,3 & BACKGROUND
1335 POKE 54279,PM:REM TELL ANTIC PMBASE
1340 POKE 1686,PM:REM POKE HI BYTE PMBASE INTO SUBROUTINE
1350 A=USR(1696,PMBASE):REM CLEAR PM MEMORY
1359 REM READ PLAYER DATA INTO STORAGE AREA
1360 FOR I=0 TO 9
1370 READ VAL:POKE PMBASE+I,VAL:NEXT I
1380 FOR I=0 TO 9
1390 READ VAL:POKE PMBASE+128+I,VAL:NEXT I
1399 REM INITILAZATION 1ST TIME ONLY
1400 Y0OLD=80:YM0OLD=10
1410 GOTO 50
10000 REM PLAYER SUBROUTINE DATA
```

```
10005 DATA 104,162,0,104,104,157,145,6,232,224,5,208,246,173,149
10006 DATA 6,174,145,6,157,0,208,173,150,6
10010 DATA 133,213,24,105,4,133,207,173,145,6,201,4,176,47,170
10015 DATA 189,133,6,133,212,202,48,4,230,207
10020 DATA 208,249,173,147,6,133,206,169,0,168,145,206,200,204
10025 DATA 146,6,144,248,173,148,6,133,206,160,0
10030 DATA 177,212,145,206,200,204,146,6,144,246,96,56,233,4,170
10035 DATA 198,207,173,147,6,133,206,160,0,177
10040 DATA 206,61,137,6,145,206,200,204,146,6,144,243,173,148,6
10045 DATA 133,206,160,0,177,206,29,141,6,145
10050 DATA 206,200,204,146,6,144,243,96,0,64,128,192,252,243
10055 DATA 207,63,3,12,48,192,0,0,0,0,0,0
10069 REM CLEAR PM AREA ROUTINE DATA
10070 DATA 104,104,133,213,104,133,212,162,0,160,0,169,0,145,212
10075 DATA 200,208,251,230,213,232,224,8,144,240,96
10105 REM PLAYER DATA
10110 DATA 153,153,189,189,255,255,189,189,153,153
10115 DATA 255,255,255,255,255,255,255,255,255,255
```

Explanation of Player-Missile Subroutine

The player-missile subroutine was designed to handle both players and missiles simultaneously. In order to do this while retaining compact and relocatable code, several compromises were made. First, only single-line player resolution is offered. We felt that the coarser, double-line resolution could be simulated by setting the player-missile stripe at double width, then double plotting each of the shape's bytes. An attempt to give you a choice of resolution modes would have created a long and messy algorithm since the memory requirements of each are quite different mathematically. In single-line resolution the actual memory storage areas for each player are exactly one page or 256 bytes apart. This makes setting the pointers to the memory area easy since only the high byte is involved. But in double-line resolution the memory areas are 128 bytes apart. A much more complicated multi-byte add is required to set the pointers.

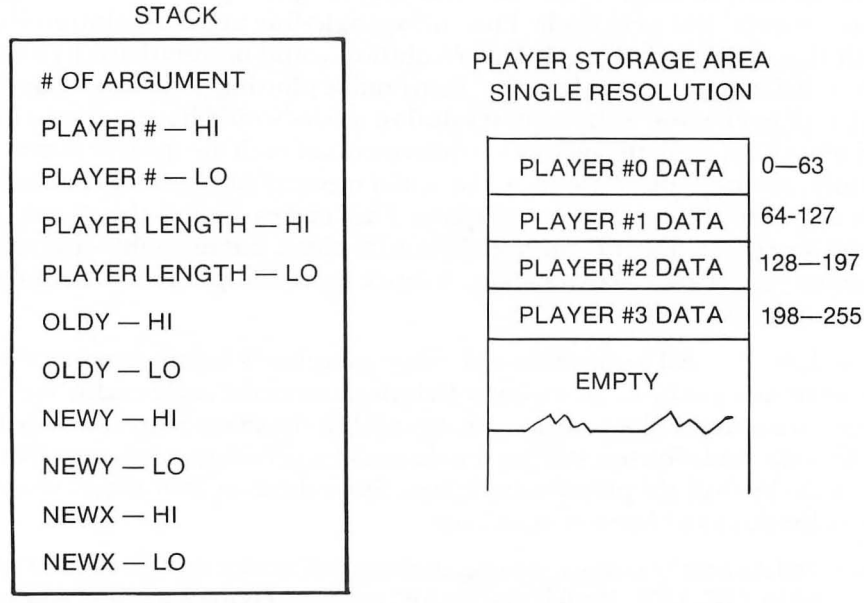
Second, we decided to erase the old shape completely before drawing the player shape in its new position. It is a faster technique since the more traditional method needs to do a series of block memory moves to shift the shape more than one line at a time. Our method requires storing the shapes in a permanent safe spot. We choose the first 256 bytes of the player-missile area. Since there are four player shapes, each shape is limited to 64 bytes or scan lines.

Advanced Assembly language programmers will notice that we did not write the subroutine in VBLANK. We felt that BASIC was slow enough without adding small delays while waiting for the VBLANK interrupt to begin our subroutine. Moving several players and their corresponding missiles on the screen at the same time could slow the game down. Motion smoothness is another reason for using VBLANK for player-missile graphics. However, as you will see in the following game examples, the animation frame rate is not fast enough to produce smooth animation with or without VBLANK.

The subroutine is divided into three parts: a routine to interpret and store the passed parameters in the USR function; a routine for erasing and drawing the player shape; and a routine for erasing and drawing a missile. Once the parameters are stored, a test on the player number determines if it is actually a missile, and if so, branches to that part of the code.

Interpreting USR Parameters

A USR function in BASIC pushes all of its parameters onto the stack before it enters the Machine language subroutine. The stack is like a dish dispenser in that the last value placed on the stack must be pulled off first. The first byte contains the number of passed parameters, a useless value, and one to be discarded. The other parameters are stored in two-byte pairs, high-byte first, in the order that you passed them. The high-bytes in our example are useless since none of our values exceeds 255. The five pairs are pulled off the stack in an indexed loop using the X register. They are pulled two bytes at a time. Only the second, or low byte is stored sequentially at PLAYNUM,X ($691, X). Thus location $691 contains the player number, $692 the player length, $693 the value for OLDY, etc.



Erasing and Storing Player Shapes

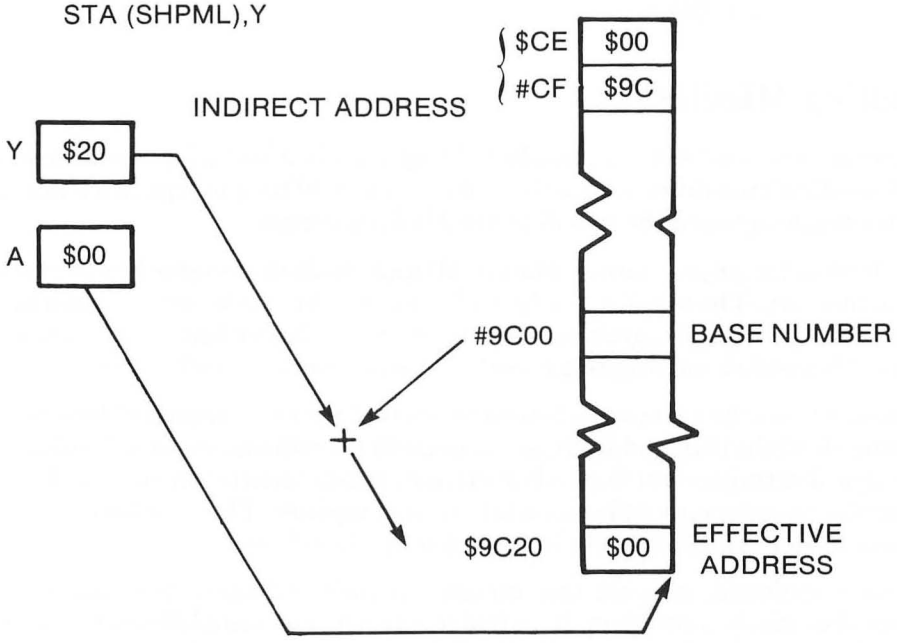
The pointers to move our player or missile shape from their storage area to the proper player-missile memory area are set up next. The shape is stored at SHAPEL, SHAPEH. The high byte BASE is just the beginning of the P/M storage area which was POKEd into location 1686 decimal ($696) from BASIC. The low byte is obtained

from a table called INDEX. Each of the values in this four-byte table are sixty-four bytes ($40) apart. The actual memory location that the shape is to be stored at in the P/M area is SHPML, SHPMH. Since each player is 256 bytes or a page apart in memory, we added #$04 to BASE in order to set it for player #0. We then did a cute little trick. We put it in a loop and incremented SHPMH for each player number.

Erasing the old player shape from memory could be handled quite easily with simple indexing in the form STA ADDRESS,Y if only one single player were involved. ADDRESS would be the absolute address of that player's P/M memory, and the Y register would contain the value YOLD. Unfortunately, the high byte of each player's P/M memory area is different. Rather than try to update ADDRESS each time, it is more efficient if the two-byte page address is stored in zero page. Then indirect index addressing in the form STA (SHPML),Y can be used either to erase or plot player-missile data.

If the computer finds a $00 in location $CE (SHPML), and a $9C in location $CF (SHPMH), then the base address is $9C00. The Y register contains the value of the OLDY position. If the shape was thirty-two scan lines down the screen, then the Y register = $20. If the computer wished to erase the shape, then it would store a #$00 in the Accumulator into memory location $9C00 + $20, or $9C20 as shown:

INDIRECT INDEX ADDRESSING



The actual code to erase the player shape is reiterated in a loop PLAYLEN times. Since it is easier to increment the Y register from zero until it is equal to PLAYLEN, the vertical screen offset is stored in SHPML in zero page. The address SHPML, SHPMH is now the address of the beginning of the player-missile shape. The Y register offsets into the shape. The code is shown below.

```
ERASE   LDA OLDY      ;Y VALUE
        STA SHPML
        LDA #$00      ;ERASE WITH 0
        TAY           ;Y REGISTER = 0
.1      STA(SHPML),Y ;STORE IN PROPER P/M AREA
        CPY PLAYLEN   ;DONE?
        BLT .1
```

The code for drawing our shape is quite similar, except that instead of storing zeros in P/M memory, we transfer the player shape from its storage area to its proper place in P/M memory. The pointers to its storage area, SHAPEL, SHAPEH were previously set up in zero page. We need update only SHPML, the low byte pointer to the place we are actually moving our player, with the NEWY position. The high byte SHPMH remains the same. The code follows:

```
        LDA NEWY      ;NEW Y VALUE
        STA SHPML     ;SETUP POINTER TO PLOT
        LDY #$00
DRAW    LDA (SHAPEL),Y;LOAD BYTE FROM PLAYER SHAPE TABLE
        STA (SHPML),Y ;STORE IN PROPER P/M PLAYER AREA
        INY           ;NEXT BYTE
        CPY PLAYLEN   ;DONE?
        BLT DRAW
```

Handling Missiles

Player numbers 4-7 refer to missiles 0-3 respectively in our player-missile subroutine. If we didn't use different numbers, the user would have to type in a letter M or P to differentiate between the two different kinds of sprites.

While missiles are just narrow players, all four reside in the same 256-byte block or page of memory. The missiles, each of which is two bits wide, are arranged parallel to each other. Essentially, all four two-bit pairs for each scan line are in the same byte of data. This makes moving one missile but not others somewhat difficult.

Missile #0 uses the first two or lowest two bits of the byte, missile #1 bits three and four, missile #2 the fifth and sixth, and missile #3 the two highest bits. While the data in one byte determines which pixels are lit for all four missiles on any scan line, each missile has an independent horizontal position register. Thus, each of the missiles can have movement completely independent of the others.

As we mentioned, moving one missile vertically without changing the other missiles' data can be a problem. If we had just two missiles initially on the same scan line, and we attempted to move missile #0 downward one scan line, either a memory move or an erase before redrawing would affect the missile #1 as well. In the first case, missile #1 would move in tandem with missile #0, while in the second case missile #1 would be erased. The solution is to mask off the other missile's bits during the erase, and to draw the missile's new position using another masking operation that will not affect the remaining bits.

ORA Instruction

This drawing technique uses the OR memory with Accumulator (ORA) instruction. It works on the bit level. If the bits in either memory or the Accumulator are on, then the result is one. If neither is on, the result is zero.

	ACCUMULATOR BIT	MEMORY BIT	RESULT BIT IN ACCUMULATOR
	0	0	0
ORA	1	0	1
	1	0	1
	1	1	1

If missile #0 was already on a particular scan line and you wanted missile #1 to be on the same scan line you would ORA missile #1 with the byte in P/M memory.

```
          0 0 0 0 0 0 1 1     MEMORY
ORA       0 0 0 0 1 1 0 0     MISSILE SHAPE
-------------------------
          0 0 0 0 1 1 1 1     RESULT
```

AND Instruction

Another selective drawing technique is the And Memory with Accumulator (AND) instruction. It too works on a bit level and is used to filter or mask out certain bits in the Accumulator. Both the memory bit and the Accumulator bits must be set (on) for the result to be one. If either memory bit is off, or both bits are off, the result is zero. We put ones where we don't want to change bit values, and zeroes where we do. We can erase one missile at a time without affecting the others.

	ACCUMULATOR BIT	MEMORY BIT	RESULT BIT IN ACCUMULATOR
	0	0	0
AND	0	1	0
	1	0	0
	1	1	1

If you wish to erase only one of the two missiles on the same scan line, you AND the data byte with a mask that always produces a zero bit result in the missile bits to be erased, and a one in all the other bits. For example, to erase missile #0 and not missile #2, the mask with which you AND the data bit is $FC.

```
          0 0 1 1 0 0 1 1      MEMORY $33
AND       1 1 1 1 1 1 0 0      MASK $FC
-------------------------
          0 0 1 1 0 0 0 0      RESULT $30
```

There are four different missile masks:

Missile mask #0	1 1 1 1 1 1 0 0	$FC
Missile mask #1	1 1 1 1 0 0 1 1	$F3
Missile mask #2	1 1 0 0 1 1 1 1	$CF
Missile mask #3	0 0 1 1 1 1 1 1	$3F

Likewise, there are four different missile shapes. For simplicity and visibility we made each missile two pixels wide. The height is controlled by the length, which is nominally two bytes. However, missiles can assume tall, thin dimensions if the user chooses a length of four or more bytes.

Missile shape #0	0 0 0 0 0 0 1 1	$03
Missile shape #1	0 0 0 0 1 1 0 0	$0C
Missile shape #2	0 0 1 1 0 0 0 0	$30
Missile shape #3	1 1 0 0 0 0 0 0	$C0

The data for the above two tables appears in our player-missile subroutine as MASKS and SHOTS.

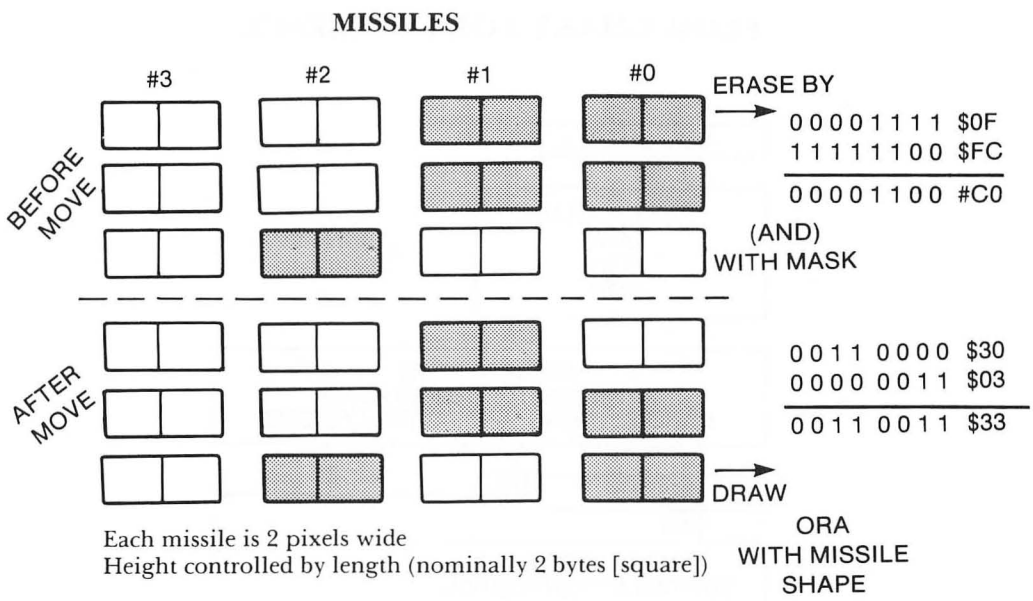
The heart of the missile routine is the erase and draw code. Each is a mere three lines long. The selected missile is erased by ANDing memory with the proper mask.

```
LDA (SHPML),Y ;LOAD OLD MISSILE DATA
AND MASKS,X   ;ERASE IT BUT DON'T DISTURB OTHER MISSILES
STA (SHPML),Y ;STORE RESULT BACK IN MEMORY
```

The selected missile redrawn in its new position with the following three lines of code.

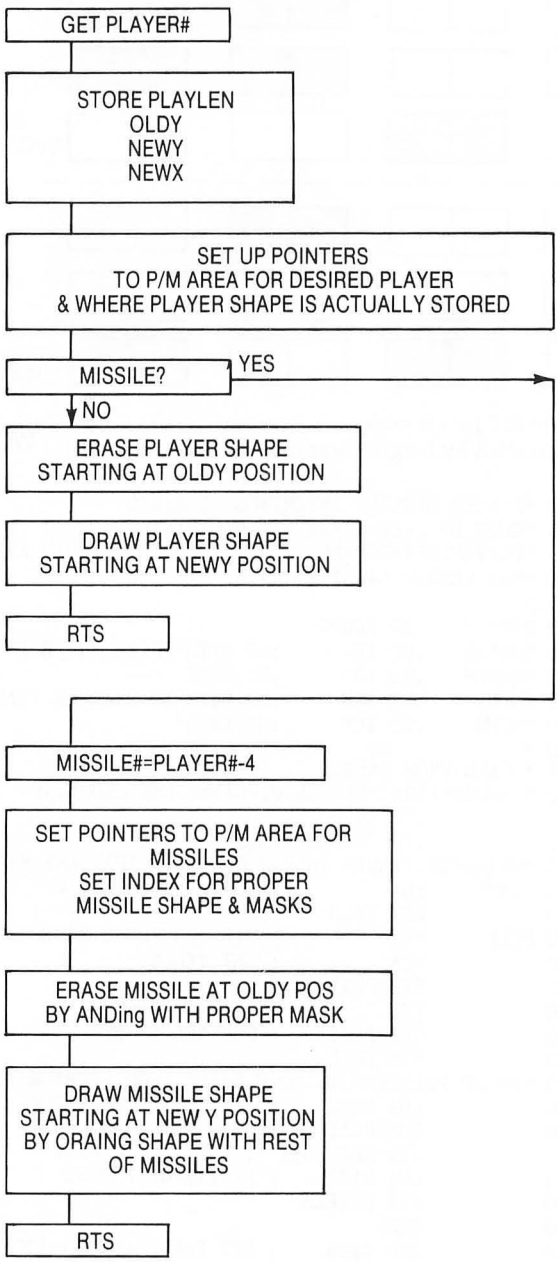
```
LDA (SHPML),Y ;LOAD OLD MISSILE DATA
ORA SHOTS,X   ;MERGE SHOT DATA WITH OTHER MISSILES
STA (SHPML),Y ;STORE NEW COMBINED MISSILE BYTE IN MEMORY
```

The technique is best illustrated with an example where we have three missiles on the screen. Missiles #0 and #1 are on the same two scan lines. Missile #2 is on the same scan line where we wish to move our missile.




```
                00010 *PLAYER MISSILE INTERFACE TO BASIC
                00015 *CODE BY JEFF STANTON & DAN PINAL
                00020 *PLAYER SHAPES ARE STORED IN PAGE ONE P/M AREA
                00030 *MAX LENGTH 64 BYTES EACH
                00035 *
D000:           00040 HPOSPO    .EQ $D000
00D4:           00050 SHAPEL    .EQ $D4      ;LO BYTE WHERE SHAPE IS STORED
00D5:           00060 SHAPEH    .EQ $D5      ;HI BYTE
00CE:           00070 SHPML     .EQ $CE      ;LO BYTE TO DRAW OR ERASE SHAPE IN P/M AREA
00CF:           00080 SHPMH     .EQ $CF      ;HI BYTE
                00090 *
                00100 * CALL FROM BASIC
                00110 * A=USR(1536, PLAYER #,PLAYER LEN,OLD Y,NEW Y,NEW X)
                00120           .OR $600
                00130           .TF "D:PMSUB.OBJ"
                00135 *TRANSFER VALUES IN USR FUNCTION THAT ARE STORED ON STACK TO SUBROUTINE
0600: 68        00150 START     PLA          ; PULL # OF ARG'S OFF STACK
0601: A2 00     00160           LDX #$00
0603: 68        00180 PULL      PLA          ; DISCARD THE HIBYTE IT'S USELESS
0604: 68        00190           PLA          ; GET VALUE
0605: 9D 91 06  00200           STA PLAYNUM,X
0608: E8        00210           INX
0609: E0 05     00220           CPX #$05     ; GOT ALL 5 YET?
060B: D0 F6     00230           BNE PULL
                00240 *SETUP POINTERS TO MOVE SHAPE FROM STORAGE PLACE TO PLACE IN P/M AREA
060D: AD 95 06  00250           LDA NEWX     ; HORIZONTAL POSITION
0610: AE 91 06  00260           LDX PLAYNUM
0613: 9D 00 D0  00270           STA HPOSPO,X  ; TELL ANTIC NEWX
0616: AD 96 06  00280           LDA BASE     ; HIBYTE OF PMBASE
0619: 85 D5     00290           STA SHAPEH
061B: 18        00300           CLC
061C: 69 04     00310           ADC #$04     ; SET FOR PLAYER 0 ADDRESS
061E: 85 CF     00320           STA SHPMH
0620: AD 91 06  00330           LDA PLAYNUM ; PLAYER #
0623: C9 04     00340           CMP #$04     ; MISSILES?
0625: B0 2F     00350           BGE MISSILES
0627: AA        00360           TAX          ;PLAYER # BECOMES INDEX
```

FLOW CHART FOR INTERFACE

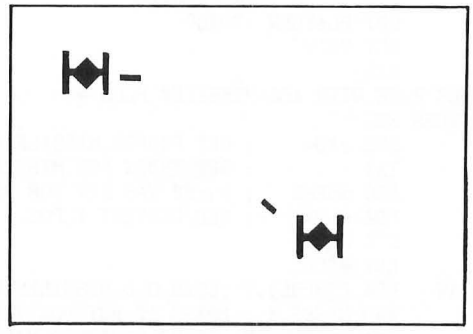


```
0628: BD 85 06 00370          LDA INDEX,X
062B: 85 D4    00380          STA SHAPEL
062D: CA       00400 .1       DEX          ;WE INCREMENT SHPMH FOR EACH PLAYER # UNLESS 0
062E: 30 04    00410          BMI ERASE
0630: E6 CF    00420          INC SHPMH    ;EACH INDIVIDUAL P/M AREA 256 BYTES APART
0632: DO F9    00430          BNE .1       ; ALWAYS
               00440 ERASE
0634: AD 93 06 00450          LDA OLDY     ; Y
0637: 85 CE    00460          STA SHPML
0639: A9 00    00470          LDA #$00     ;ERASE WITH 0
063B: A8       00480          TAY          ; Y=0
063C: 91 CE    00490 .1       STA (SHPML),Y ;STORE IN PROPER P/M AREA
063E: C8       00500          INY          ;NEXT BYTE
063F: CC 92 06 00510          CPY PLAYLEN ; DONE?
0642: 90 F8    00520          BLT .1
0644: AD 94 06 00530          LDA NEWY     ; NEW Y
0647: 85 CE    00540          STA SHPML    ;SETUP POINTER TO PLOT
0649: AO 00    00550          LDY #$00
064B: B1 D4    00570 DRAW     LDA (SHAPEL),Y ;LOAD BYTE FROM PLAYER SHAPE TABLE
064D: 91 CE    00580          STA (SHPML),Y ;STORE IN PROPER P/M PLAYER AREA
064F: C8       00590          INY          ;NEXT BYTE
0650: CC 92 06 00600          CPY PLAYLEN ;DONE?
0653: 90 F6    00610          BLT DRAW
0655: 60       00620          RTS
               00650 *COMES HERE WITH ACC=MISSILE# PLUS 4
0656: 38       00660 MISSILES SEC
0657: E9 04    00670          SBC #$04     ; GET PROPER MISSILE # (0-3)
0659: AA       00680          TAX          ; SET INDEX FOR MISSILE MASKS
065A: C6 CF    00690          DEC SHPMH    ; SHPMH WAS SET FOR PLAYER0, MISSILES ARE 1
065C: AD 93 06 00700          LDA OLDY     ; OLD/CURRENT Y POS.          \PAGE LOWER
065F: 85 CE    00710          STA SHPML
0661: AO 00    00720          LDY #$00
0663: B1 CE    00740 UNDRAW   LDA (SHPML),Y ;LOAD OLD MISSILES DATA
0665: 3D 89 06 00750          AND MASKS,X ; ERASE IT BUT DON'T DISTURB OTHER MISSILES
0668: 91 CE    00760          STA (SHPML),Y ;STORE COMBINED MISSILE DATA IN P/M AREA
066A: C8       00770          INY          ;NEXT BYTE
066B: CC 92 06 00780          CPY PLAYLEN ; DONE?
066E: 90 F3    00790          BLT UNDRAW
0670: AD 94 06 00800          LDA NEWY     ; NEW Y POS.
0673: 85 CE    00810          STA SHPML
0675: AO 00    00820          LDY #$00
0677: B1 CE    00840 MDRAW    LDA (SHPML),Y ;LOAD OLD MISSILES DATA
0679: 1D 8D 06 00850          ORA SHOTS,X ; MERGE SHOT DATA WITH OTHER MISSILES
067C: 91 CE    00860          STA (SHPML),Y ;STORE NEW COMBINED MISSILE BYTE
067E: C8       00870          INY          ;NEXT BYTE
067F: CC 92 06 00880          CPY PLAYLEN ;DONE?
0682: 90 F3    00890          BLT MDRAW
0684: 60       00900          RTS
               00910 ;
0685: 00 40 80
0688: C0       00920 INDEX    .HS 004080C0 ;LO BYTE OF STORED PLAYER SHAPES
0689: FC F3 CF
068C: 3F       00930 MASKS    .HS FCF3CF3F ;MISSILE MASKS
068D: 03 0C 30
0690: C0       00940 SHOTS    .HS 030C30C0 ;MISSILE SHAPES
0691:          00950 PLAYNUM  .BS 1
0692:          00960 PLAYLEN  .BS 1
0693:          00970 OLDY     .BS 1
0694:          00980 NEWY     .BS 1
0695:          00990 NEWX     .BS 1
0696:          01000 BASE     .BS 1        ;HI BYTE PMBASE IS BOKED IN FROM BASIC
```

Two Ship Example

We can write a simple two-player shoot-'em-up game in BASIC if we use our player-missile subroutine to provide enough animation frame or speed for playability. The code for the second ship is nearly identical to that for the first ship. In fact, with the exception of using variables having a 1 at the tail end of the variable names, the code is the same, line for line. The code for player #2 follows the code for player #1.

The code for each player includes: a joystick read routine to determine the ship's new velocity vector and to update its position; a similar routine to determine the missile's velocity and direction; and logic to prevent either the ship or the missile from leaving the screen boundaries. The ship in our previous example stopped dead while the missile moved. These ships, however, not only continue in their present course during the missile flight; they even alter course to evade enemy missiles.

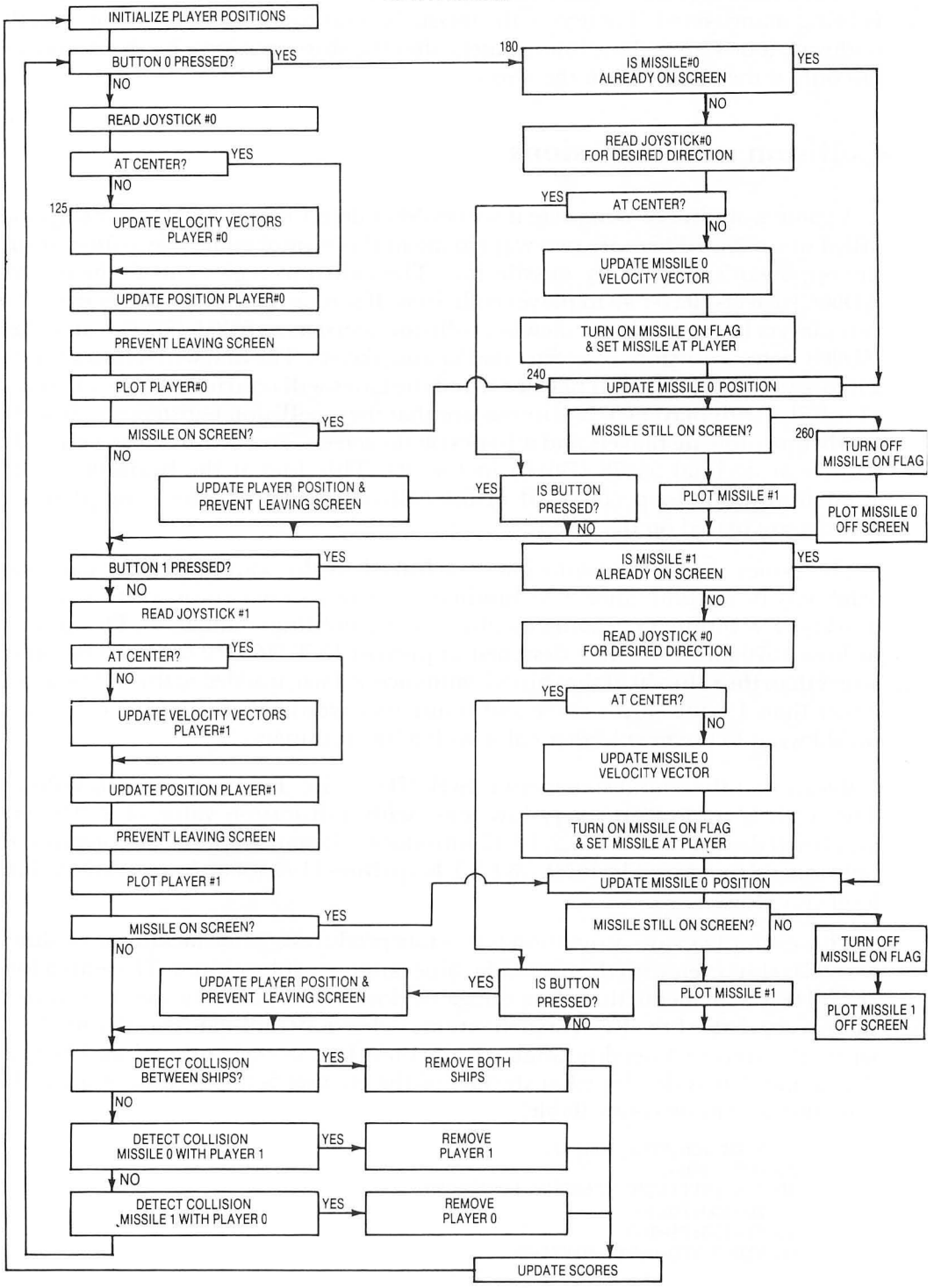


In the previous example, if the missile movement code was executed, it branched past the player movement code. In this example, the missile code updates the player position, and the player position code updates the missile position. This enables the player to continue moving or maneuvering while the missile is in flight.

If you look at the game's flow chart for each player, you will notice that the two possible paths are decided by whether the joystick button is pressed. Pressing the button fires a missile in the desired direction, only if there isn't another missile already on the screen. Of course, if you don't give it a direction (joystick centered), it skips firing the missile and just updates the ship's position based on its current heading. When it fires the missile, it turns the missile flag on (M0FLAG = 1). To prevent it from firing again before the missile reaches the edge of the playfield, the program tests M0FLAG in line 430. If the flag is on, it just updates the missile's position based on its current trajectory. Finally, once the missile reaches the screen edge, it turns the flag off (M0FLAG = 0) and plots the missile offscreen.

When the joystick button isn't pressed, the program code reaches line 110, the beginning of the joystick read routine which determines the ship's new velocity vector. This enables the ship to change direction. Thus, if a player wishes to change direction immediately upon firing his missile, he must release the button quickly, so that it reaches this code on the next animation frame.

ATTACK


INITIALIZE PLAYER POSITIONS
BUTTON 0 PRESSED?
YES
NO
READ JOYSTICK #0
AT CENTER?
YES
NO
125
UPDATE VELOCITY VECTORS PLAYER #0
UPDATE POSITION PLAYER#0
PREVENT LEAVING SCREEN
PLOT PLAYER#0
MISSILE ON SCREEN?
YES
NO
UPDATE PLAYER POSITION & PREVENT LEAVING SCREEN
YES
IS BUTTON PRESSED?
NO
BUTTON 1 PRESSED?
YES
NO
READ JOYSTICK #1
AT CENTER?
YES
NO
UPDATE VELOCITY VECTORS PLAYER#1
UPDATE POSITION PLAYER#1
PREVENT LEAVING SCREEN
PLOT PLAYER #1
MISSILE ON SCREEN?
YES
NO
UPDATE PLAYER POSITION & PREVENT LEAVING SCREEN
YES
IS BUTTON PRESSED?
NO
DETECT COLLISION BETWEEN SHIPS?
YES
REMOVE BOTH SHIPS
NO
DETECT COLLISION MISSILE 0 WITH PLAYER 1
YES
REMOVE PLAYER 1
NO
DETECT COLLISION MISSILE 1 WITH PLAYER 0
YES
REMOVE PLAYER 0
UPDATE SCORES
180
IS MISSILE#0 ALREADY ON SCREEN
YES
NO
READ JOYSTICK#0 FOR DESIRED DIRECTION
YES
AT CENTER?
NO
UPDATE MISSILE 0 VELOCITY VECTOR
TURN ON MISSILE ON FLAG & SET MISSILE AT PLAYER
240
UPDATE MISSILE 0 POSITION
MISSILE STILL ON SCREEN?
260
TURN OFF MISSILE ON FLAG
PLOT MISSILE #1
PLOT MISSILE 0 OFF SCREEN
IS MISSILE #1 ALREADY ON SCREEN
YES
NO
READ JOYSTICK #0 FOR DESIRED DIRECTION
YES
AT CENTER?
NO
UPDATE MISSILE 0 VELOCITY VECTOR
TURN ON MISSILE ON FLAG & SET MISSILE AT PLAYER
UPDATE MISSILE 0 POSITION
MISSILE STILL ON SCREEN?
NO
TURN OFF MISSILE ON FLAG
PLOT MISSILE #1
PLOT MISSILE 1 OFF SCREEN

Once a missile has been launched, it will continue on its path even while the ship is being maneuvered. Therefore, the missile's position needs to be updated in this pathway, too. This is done immediately after the ship's position has been updated, but only if the missile is on the screen.

Collision and Explosions

A game wouldn't be complete if we couldn't detect if one or the other ship were killed in combat. There are two ways to die in this type of game—by collision with the opponent's ship or by missile fire. The collision register at decimal 53260 ($D00C) detects player #0 to player collisions. If it returns a value greater than zero, two players have collided. Likewise, collision registers at 53256 ($D008) and 53257 ($D009) detect collisions between missiles and players. The first will return the value 2 if missile #0 collides with player #1, and the latter will return the value 1 if missile #1 collides with player #0. It is important that these collision registers are cleared to zero before plotting players and missiles on the screen. You do this via the HITCLR register at decimal 53278 (D01E) in line 90. This line at the beginning of the animation frame loop clears all of the collision registers before any players or missiles are placed on the screen.

The rather simplistic explosions are linked to the sound routine. Each ship brightens from luminance 4 to luminance 14 in its own color, then blacks out quickly. The luminance changes within a low, rumbling sound loop. The formulas in lines 1570 and 1580 were designed to prevent INT (10–I*0.66) from becoming larger than the value 10. If the ship's luminance of 4 when added to this value became larger than 15, the ship's color and luminance would change as the color value would wrap to the next higher color with a low luminance.

Recall that the sound statement is SOUND (Voice, Pitch, Distortion, Volume). The pitch is set to 250, a very low tone, with a distortion value of 4. The even numbered distortion levels 0,2,4,8,12 introduce different amounts of noise into the pure tones, 10 and 14. The FOR...NEXT loop (lines 1150-1580) decreases the volume level very slowly.

This example's slow animation frame rate produces a game lacking smoothness. BASIC is slow even with the use of Machine language subroutines. There are a lot of IF...THEN statements that slow the game down. The game, however, will run a little faster if all of the REM statements are deleted. Arcade games really need to be written entirely in Assembly language to achieve fast, smooth animation. The *Space War* game that is developed at the end of the chapter is a very similar game, but smoother and more controllable.

```
10 POKE 106,PEEK(106)-12
20 GOTO 1860
30 REM INITILIZE STARTING POSITIONS
40 XO=100:YO=80
50 X1=150:Y1=160
60 VXO=0:VYO=0:VX1=0:VY1=0
```

```
70 MOFLAG=0:M1FLAG=0
80 REM PLAYER #0
90 POKE 53278,0:REM CLEAR COLLISION REGISTER
100 IF STRIG(0)=0 THEN 430
110 Z0=STICK(0)
120 IF Z0=15 THEN 230
130 REM UPDATE VELOCITY VECTORS
140 IF Z0=14 THEN VX0=0:VY0=-3
150 IF Z0=13 THEN VX0=0:VY0=3
160 IF Z0=10 THEN VX0=-2:VY0=-3
170 IF Z0=9 THEN VX0=-2:VY0=3
180 IF Z0=11 THEN VX0=-2:VY0=0
190 IF Z0=7 THEN VX0=2:VY1=0
200 IF Z0=6 THEN VX0=2:VY0=-3
210 IF Z0=5 THEN VX0=2:VY0=3
220 REM UPDATE POSITION PLAYER0
230 X0=X0+VX0:Y0=Y0+VY0
240 IF X0<46 THEN X0=46
250 IF X0>198 THEN X0=198
260 IF Y0<24 THEN Y0=24
270 IF Y0>216 THEN Y0=216
280 A=USR(1536,0,10,Y0OLD,Y0,X0)
290 Y0OLD=Y0
300 IF M0FLAG=0 THEN 780
310 REM UPDATE POSITION MISSILE 0
320 XM0=XM0+VXM0:YM0=YM0+VYM0
330 IF XM0<40 THEN 400
340 IF XM0>207 THEN 400
350 IF YM0<20 THEN 400
360 IF YM0>220 THEN 400
370 A=USR(1536,4,2,YM0OLD,YM0,XM0)
380 YM0OLD=YM0:GOTO 420
390 REM REMOVE MISSILE TO OFF SCREEN
400 M0FLAG=0
410 A=USR(1536,4,2,YM0OLD,YM0OLD,10)
420 GOTO 780
430 IF M0FLAG=1 THEN 590
440 Z0=STICK(0)
450 IF Z0=15 THEN 690
460 IF Z0=14 THEN VXM0=0:VYM0=-6
470 IF Z0=13 THEN VXM0=0:VYM0=6
480 IF Z0=10 THEN VXM0=-5:VYM0=-6
490 IF Z0=9 THEN VXM0=-5:VYM0=6
500 IF Z0=11 THEN VXM0=-5:VYM0=0
510 IF Z0=7 THEN VXM0=5:VYM0=0
520 IF Z0=6 THEN VXM0=5:VYM0=-6
530 IF Z0=5 THEN VXM0=5:VYM0=6
540 M0FLAG=1
550 XM0=X0+4:YM0=Y0+2
560 FOR I=15 TO 0 STEP -0.25
570 SOUND 0,10,0,I:NEXT I
580 REM UPDATE POSITION MISSILE 0
590 XM0=XM0+VXM0:YM0=YM0+VYM0
600 IF XM0<40 THEN 670
610 IF XM0>207 THEN 670
620 IF YM0<20 THEN 670
630 IF YM0>220 THEN 670
640 A=USR(1536,4,2,YM0OLD,YM0,XM0)
650 YM0OLD=YM0:GOTO 700
660 REM REMOVE MISSILE TO OFF SCREEN
670 M0FLAG=0
```

```
680 A=USR(1536,4,2,YMOOLD,YMOOLD,10)
690 REM UPDATE POSITION PLAYER0
700 X0=X0+VX0:Y0=Y0+VY0
710 IF X0<46 THEN X0=46
720 IF X0>198 THEN X0=198
730 IF Y0<24 THEN Y0=24
740 IF Y0>216 THEN Y0=216
750 A=USR(1536,0,10,Y00LD,Y0,X0)
760 Y00LD=Y0
770 REM PLAYER #1
780 IF STRIG(1)=0 THEN 1110
790 Z1=STICK(1)
800 IF Z1=15 THEN 910
810 REM UPDATE VELOCITY VECTORS
820 IF Z1=14 THEN VX1=0:VY1=-3
830 IF Z1=13 THEN VX1=0:VY1=3
840 IF Z1=10 THEN VX1=-2:VY1=-3
850 IF Z1=9 THEN VX1=-2:VY1=3
860 IF Z1=11 THEN VX1=-2:VY1=0
870 IF Z1=7 THEN VX1=2:VY1=0
880 IF Z1=6 THEN VX1=2:VY1=-3
890 IF Z1=5 THEN VX1=2:VY1=3
900 REM UPDATE POSITION PLAYER1
910 X1=X1+VX1:Y1=Y1+VY1
920 IF X1<46 THEN X1=46
930 IF X1>198 THEN X1=198
940 IF Y1<24 THEN Y1=24
950 IF Y1>216 THEN Y1=216
960 A=USR(1536,1,10,Y10LD,Y1,X1)
970 Y10LD=Y1
980 REM UPDATE POSITION MISSILE 1
990 XM1=XM1+VXM1:YM1=YM1+VYM1
1000 IF XM1<40 THEN 1070
1010 IF XM1>207 THEN 1070
1020 IF YM1<20 THEN 1070
1030 IF YM1>220 THEN 1070
1040 A=USR(1536,5,2,YM10LD,YM1,XM1)
1050 YM10LD=YM1:GOTO 1100
1060 REM REMOVE MISSILE TO OFF SCREEN
1070 M1FLAG=0
1080 A=USR(1536,5,2,YM10LD,YM10LD,10)
1090 IF M1FLAG=0 THEN 1470
1100 GOTO 1270
1110 IF M1FLAG=1 THEN 1270
1120 Z1=STICK(1)
1130 IF Z1=15 THEN 1380
1140 IF Z1=14 THEN VXM1=0:VYM1=-6
1150 IF Z1=13 THEN VXM1=0:VYM1=6
1160 IF Z1=10 THEN VXM1=-5:VYM1=-6
1170 IF Z1=9 THEN VXM1=-5:VYM1=0
1180 IF Z1=11 THEN VXM1=-5:VYM1=0
1190 IF Z1=7 THEN VXM1=5:VYM1=0
1200 IF Z1=6 THEN VXM1=5:VYM1=-6
1210 IF Z1=5 THEN VXM1=5:VYM1=6
1220 M1FLAG=1
1230 XM1=X1+4:YM1=Y1+2
1240 FOR I=15 TO 0 STEP -0.25
1250 SOUND 1,10,0,I:NEXT I
1260 REM UPDATE POSITION MISSILE 1
1270 XM1=XM1+VXM1:YM1=YM1+VYM1
1280 IF XM1<40 THEN 1350
```

```
1290 IF XM1>207 THEN 1350
1300 IF YM1<20 THEN 1350
1310 IF YM1>220 THEN 1350
1320 A=USR(1536,5,2,YM1OLD,YM1,XM1)
1330 YM1OLD=YM1:GOTO 1370
1340 REM REMOVE MISSILE TO OFF SCREEN
1350 M1FLAG=0
1360 A=USR(1536,5,2,YM1OLD,YM1OLD,10)
1370 IF STRIG(1)=1 THEN 1470
1380 REM UPDATE POSITION PLAYER1
1390 X1=X1+VX1:Y1=Y1+VY1
1400 IF X1<46 THEN X1=46
1410 IF X1>198 THEN X1=198
1420 IF Y1<24 THEN Y1=24
1430 IF Y1>216 THEN Y1=216
1440 A=USR(1536,1,10,Y1OLD,Y1,X1)
1450 Y1OLD=Y1
1460 REM DETECT COLLISIONS BETWEEN SHIPS
1470 IF PEEK(53260)>0 THEN 1530
1480 REM DETECT MISSILE 0 COLLISION PLAYER 1
1490 IF PEEK(53256)=2 THEN 1750
1500 REM DETECT MISSILE 1 COLLISION PLAYER 0
1510 IF PEEK(53257)=1 THEN 1640
1520 GOTO 90
1530 REM REMOVE SHIPS #0&1 IN A DUAL SHIP COLLISION
1540 X1=10:X0=10
1550 FOR I=15 TO 0 STEP -0.2
1560 SOUND 0,250,4,I:FOR W=1 TO 5:NEXT W
1570 POKE 704,148+INT(10-I*0.66)
1580 POKE 705,52+INT(10-I*0.66):NEXT I
1590 A=USR(1536,0,10,Y0OLD,Y0OLD,X0)
1600 A=USR(1536,1,10,Y1OLD,Y1OLD,X1)
1610 A=USR(1536,4,2,YM0OLD,YM0OLD,10)
1620 FOR DE=1 TO 200:NEXT DE
1630 GOTO 40
1640 REM REMOVE SHIP #0
1650 FOR I=15 TO 0 STEP -0.2
1660 SOUND 0,250,4,I:FOR W=1 TO 5:NEXT W
1670 POKE 704,148+INT(10-I*0.66):NEXT I
1680 X0=10
1690 A=USR(1536,0,10,Y0OLD,Y0OLD,X0)
1700 A=USR(1536,5,2,YM1OLD,YM1OLD,10)
1710 A=USR(1536,4,2,YM0OLD,YM0OLD,10)
1720 FOR DE=1 TO 200:NEXT DE
1730 POKE 704,152
1740 GOTO 40
1750 REM REMOVE SHIP #1
1760 FOR I=15 TO 0 STEP -0.2
1770 SOUND 0,250,4,I:FOR W=1 TO 5:NEXT W
1780 POKE 705,52+INT(10-I*0.66):NEXT I
1790 X1=10
1800 A=USR(1536,1,10,Y1OLD,Y1OLD,X1)
1810 A=USR(1536,4,2,YM0OLD,YM0OLD,10)
1820 A=USR(1536,5,2,YM1OLD,YM1OLD,10)
1830 FOR DE=1 TO 200:NEXT DE
1840 POKE 705,56
1850 GOTO 40
1860 CB=PEEK(106):CHRSET=CB*256
1870 PM=PEEK(106)+4:PMBASE=PM*256
1880 GRAPHICS 1+16
1890 REM POKE IN PM ROUTINE
```

```
1900 FOR I=0 TO 150:READ X:POKE 1536+I,X:NEXT I
1910 REM POKE IN PM CLEAR ROUTINE
1920 FOR I=0 TO 25:READ X:POKE 1696+I,X:NEXT I
1930 REM ACTIVATE P/M
1940 SETCOLOR 2,0,0
1950 POKE 559,62:REM SET DMACTL -SINGLE LINE
1960 POKE 53277,3:REM SET GRACTL -PLAYERS&MISSILES
1970 A=USR(1696,PMBASE)
1980 POKE 53256,0:POKE 53257,0:REM PLAYERS REGULAR WIDTH
1990 POKE 53260,0:REM MISSILES REGULAR WIDTH
2000 POKE 704,152:REM PLAYER0 BLUE GREEN LUM 8
2010 POKE 705,56:REM PLAYER1 RED ORANGE LUM 8
2020 POKE 54279,PM:REM TELL ANTIC PMBASE
2030 POKE 1686,PM:REM POKE HI BYTE PMBASE INTO SUBROUTINE
2040 REM READ PLAYER DATA INTO STORAGE AREA
2050 FOR I=0 TO 9
2060 READ VAL:POKE PMBASE+I,VAL:NEXT I
2070 FOR I=0 TO 9
2080 READ VAL:POKE PMBASE+64+I,VAL:NEXT I
2090 REM INITILAZATION 1ST TIME ONLY
2100 Y0OLD=80:Y1OLD=160:YM0OLD=10:YM1OLD=10
2110 GOTO 40
2120 REM PLAYER SUBROUTINE DATA
2130 DATA 104,162,0,104,104,157,145,6,232,224,5,208,246,173,149
2135 DATA 6,174,145,6,157,0,208,173,150,6
2140 DATA 133,213,24,105,4,133,207,173,145,6,201,4,176,47,170
2145 DATA 189,133,6,133,212,202,48,4,230,207
2150 DATA 208,249,173,147,6,133,206,169,0,168,145,206,200,204
2155 DATA 146,6,144,248,173,148,6,133,206,160,0
2160 DATA 177,212,145,206,200,204,146,6,144,246,96,56,233,4
2165 DATA 170,198,207,173,147,6,133,206,160,0,1
2170 DATA 206,61,137,6,145,206,200,204,146,6,144,243,173,148
2175 DATA 6,133,206,160,0,177,206,29,141,6,145
2180 DATA 206,200,204,146,6,144,243,96,0,64,128,192,252,243
2185 DATA 207,63,3,12,48,192,0,0,0,0,0,0
2190 REM CLEAR PM AREA ROUTINE DATA
2200 DATA 104,104,133,213,104,133,212,162,0,160,0,169,0,145
2205 DATA 212,200,208,251,230,213,232,224,8,144,240,96
2210 REM PLAYER DATA
2220 DATA 153,153,189,189,255,255,189,189,153,153
2230 DATA 153,153,189,189,255,255,189,189,153,153
```

Shoot Bricks Game

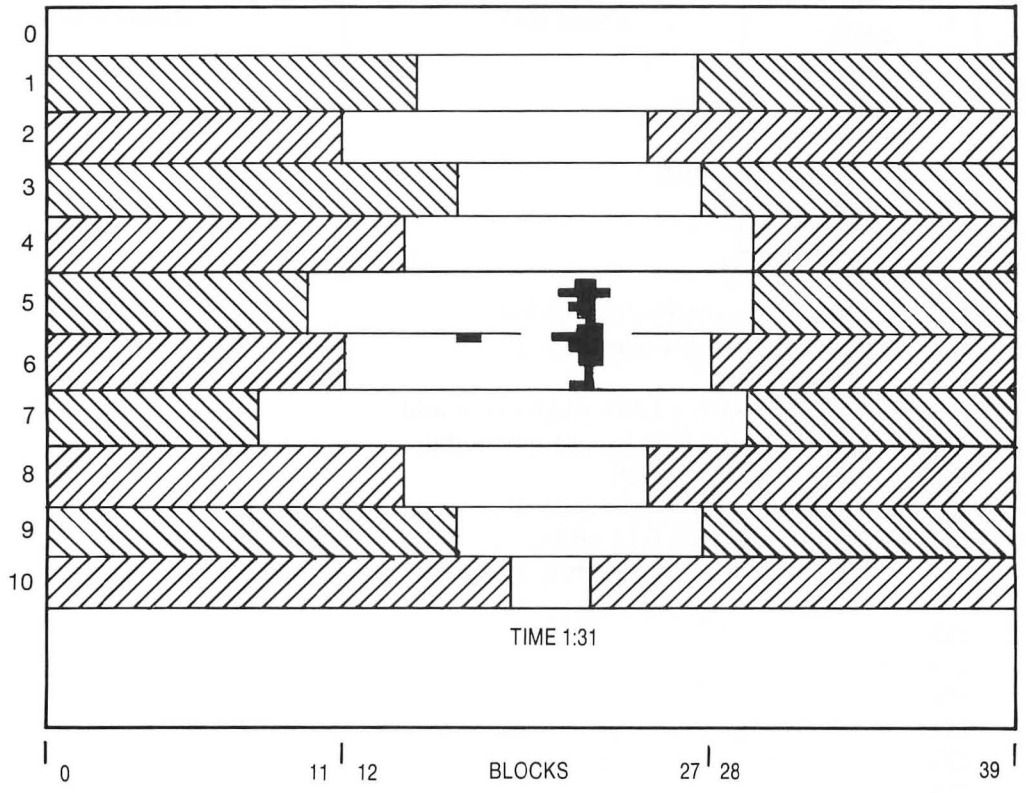
The next example uses both playfield and player-missile graphics in a timed game in which the object is to survive the longest between two crushing brick walls. The joystick-manueverable player can use a pistol to shoot any of the bricks out in the ten rows on either side of him. The bricks are replenished randomly at a rate slightly faster than the player can remove them. Thus, it becomes a matter of strategy and endurance to last for any reasonable length of time.

When designing a game like this, you must anticipate the player's strategy. The player, when confronted with the impossiblity of keeping the entire wall back, will retreat to either the top or bottom and shoot just at the blocks immediately surrounding him. If the bricks were still placed on the screen randomly even after a row of bricks closed completely, the player would have plenty of time to shoot at the few

random blocks appearing in the rows adjacent to him. However, if the random blocks are not put on the screen in the already closed rows, bricks would appear more rapidly in the few open rows. This makes the game fast-paced and somewhat challenging. The game itself has little play depth, so don't expect it to hold your interest as a game. It was primarily designed to teach programming technique.

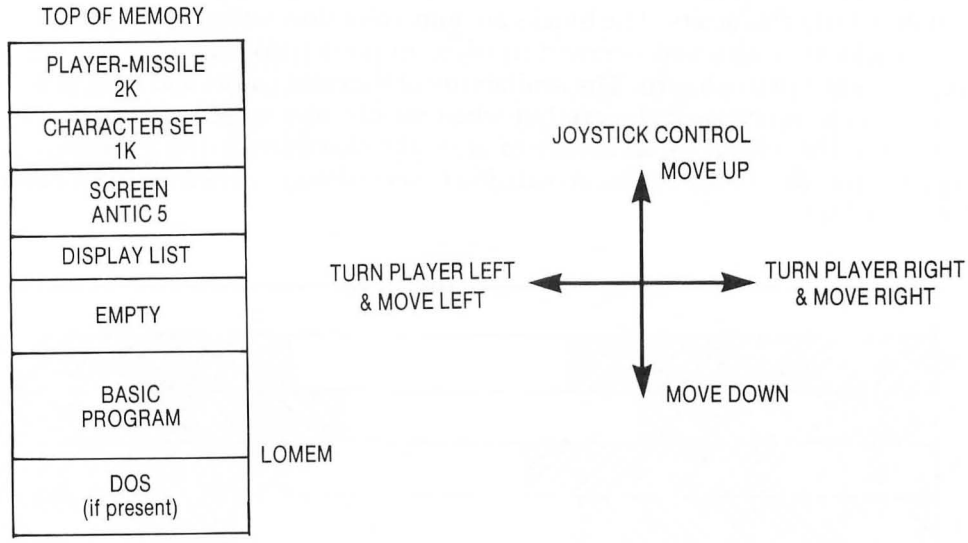
It is desirable to have a different color for each of the ten rows of bricks, but there is a maximum of only four color registers in the non-GTIA graphics modes. Therefore, we use display list interrupts to change a single color register while ANTIC is drawing the screen. It would be easier to use GTIA mode 11, but, unfortunately, this mode does not support collision registers.

We choose to do the display in ANTIC 5, one of the non-BASIC graphics modes, because the four-color characters are sixteen scan lines high. Each of the lines contains forty characters. The blocks are four color dots wide by 16 scan lines tall. The bricks are added and removed in adjacent pairs (two characters) so that they appear to be square-shaped. The availability of the extra colors had little to do with the initial programming design, but when we put in a scorekeeping timer at the bottom of the screen, we were able to draw the characters using a different color register from that of the blocks. A collision is never detected when the player touches the score line.



Setting Up Display List

The screen resides just below the top of memory which has been lowered twelve pages to make room for both the player-missile memory and new character set. We choose to modify the display list for a Graphics 1 screen because the display memory (20 columns x 24 rows) is the same as ANTIC 5 (40 columns x 12 rows), and our display list is slightly shorter. The start of the display list, which is shadowed at locations 560 and 561, is exactly the same for both graphics modes. Therefore, it is easy to POKE in our new display list at DLIST = PEEK(560)+PEEK(561)*256. This value is 37216 for 48K machines. If we substitute the following 20-byte display list for the original, we will have an ANTIC 5 screen.




```
112     These three instructions print
112     24 blank scan lines at the top
112     of the screen
 69 )   ANTIC 5 with a LMS instruction added
128 }   Address of the first line of screen data
145 )   145*256 +128 = 37248
133     Display the rest of the data in
133     ANTIC 5 with a DLI added
133     We have a total of 11 Antic 5 lines
133
133
133
133
133
133
133
```

```
  7     Text mode 2 for timer display
 65 }   Jump and wait for vertical blank
 96 }   Address of vertical display list itself
145 }   145*256 + 96 = 37216
        (return to the top of this list)
```

Initialization

The initialization for this game resembles our previous example. The player-missile, clear memory, and display list interrupt subroutines are each POKEd into memory in their appropriate places in page 6. The code is stored as DATA statements. The first half of the character set (512 bytes) is then moved to its new location, just above the top of memory, starting at location CHRSET which is equal to PEEK(106)*256. Only the first two characters are used to generate the screen. The 0th character is a blank and is used where there are no blocks. This includes our empty 0th row at the very top. The first character is rewritten as a solid shape using color register 1. The bit pattern for each line in the character is 1 0 1 0 1 0 1 0 decimal 170 or $AA. Using this bit pattern for color register 1 is a deliberate choice. The characters in the score line use color register 2. Thus, a collision with our score letters and numbers (playfield 2) will not produce the same value as a collision with a block (playfield 1).

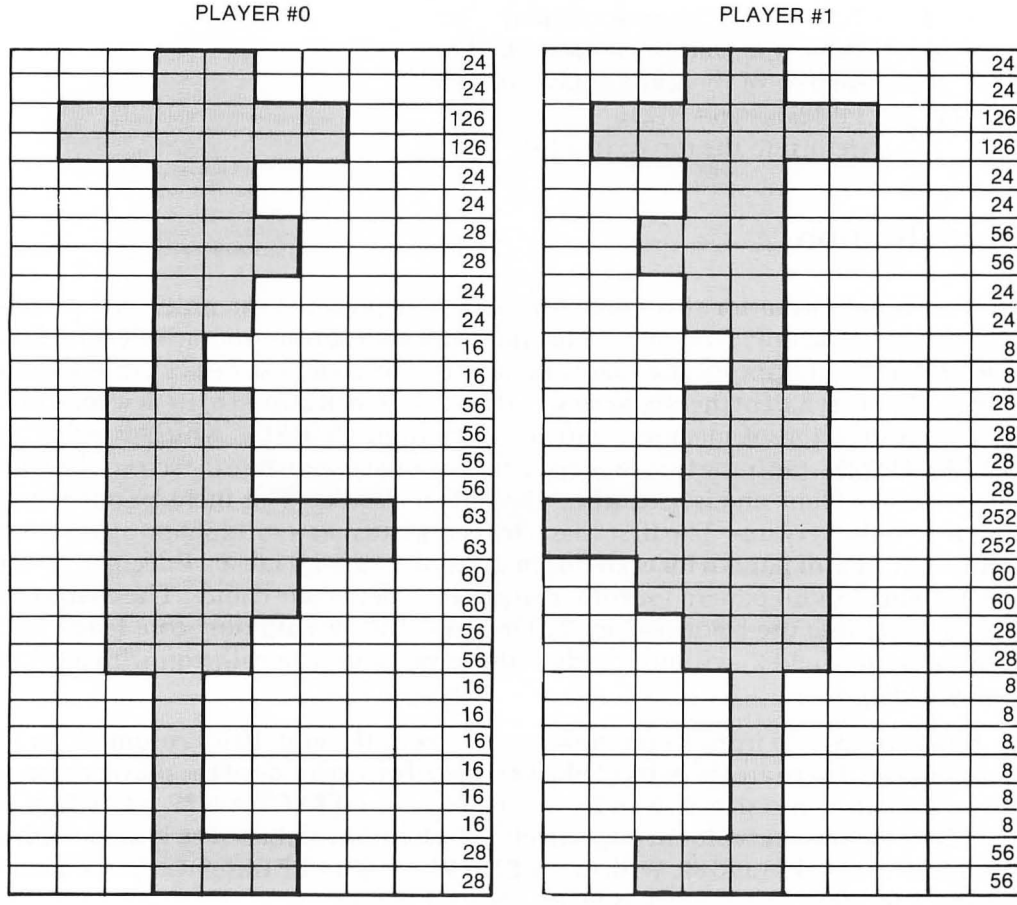
Blocks are placed initially on the screen in rows 1 through 10 for columns 0 to 11 and 28 to 39. There are six pairs of blocks situated on each side of the player on each row. The offset into screen memory for any block is OFFSET = 40*R + C, where R and C are the row and column respectively. The location of the screen is shadowed at locations decimal 88 and 89. With SCREEN = PEEK(88) + PEEK(89)*256 , the actual memory location of any block is SCREEN + OFFSET.

Player-missile graphics appear to be double-line resolution set at double width, but they are actually single-line resolution with each byte doubled. Two different players are used for the man. Player #0 faces left, and player #1 faces right. They are the same color and have the same vertical position. Player #1 is placed on the screen initially at X=125, Y=50.

The two levels of difficulty are selected with the SELECT key. It toggles between putting blocks on the screen at one-half second intervals and one second intervals. An asterisk (*) at the bottom right of the screen denotes the easier level. The START key starts the game.

Main Game Loop

The game loop tests when to place a block randomly on the screen at one-half or one second intervals. VBLANK increments the timer at decimal 20 every sixtieth of a second. When it overflows (TIMER >255), location 19 is incremented. If we set TIMER = 195, after sixty cycles occur (one second), location 19 becomes 1. This



makes a convenient test to determine when one second has elapsed. Likewise if TIMER is set to 225 with the SELECT key, after one-half second location 19 would be incremented. The test at line 210, IF PEEK(19)=0 THEN 344, skips putting a new block on the screen and updating the scoring timer unless the proper timing interval has elapsed.

There are two pointer arrays, L(10) and R(10), that keep track of the inside wall boundaries closest to the player for each row. Initially each of the L(I) elements are set to 11 and each of the R(I) elements are set to 28. As blocks are added and subtracted, these values begin to change by multiples of two. For example, if a block were added to both the left and right sides of row 3, then L(3)=13 and R(3)=26. If blocks were added just to the left side, eventually the sides would touch when L(3)=25 and R(3)=26. Since we don't want to add a block to a row that is already touching, we could test if R(I)-L(I)=1. If it equals 1, then we choose a different random row and random side to try to place another block. Eventually, we find a place even if it is in one of the rows to the right or left of our player.

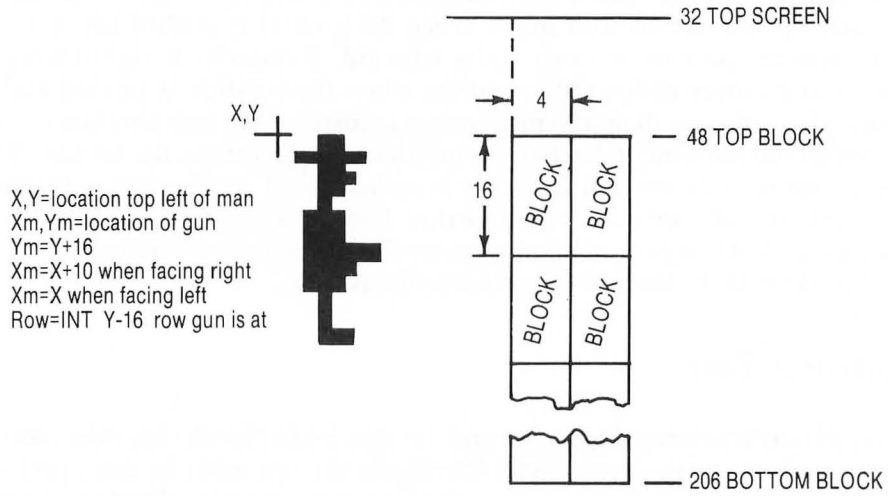
The gun fighter can be maneuvered around the vacant area of the playfield by a

player using a joystick. When the joystick is positioned up or down, the man moves vertically up or down by four units. When the joystick is pushed left, a left-facing figure appears and moves two units leftward. Similarly, a right-facing figure appears and moves rightward two units when the joystick is pushed right. The routine also supports diagonal movements incorporating combinations of vertical and horizontal movement. Each of the joystick movements sets the variable PLAY to zero or one to indicate which player is to be placed on the screen through our player-missile subroutine. The player that does not appear on the screen is placed offscreen at X=10. Again remember to set the old Y position equal to the new Y position (YOLD=Y) just after the subroutine is used.

Collision Test

A collision between either player and the playfield #1 blocks has to be tested in two different places in the game code. Obviously the test must be done just after the player moves, but it also has to be done just after a new random block is placed on the screen. A collision is detected in either case if the value 2 is set in player #0's collision register 53252 ($D004) or player #1's collision register 53253 ($D005). Since the bricks appear to be crushing our man, it is effective to squash him by setting the player width back to normal. When this happens, it appears that the man is compressed towards the left. This occurs because the player image is always plotted from left to right and begins at the value in the horizontal position register. The GTIA just double plots player pixels when set to double width. The player struck by the left wall remains in contact with the wall when it is compressed, but the player struck by the right wall will shift to the left, or away from the wall, eight pixels. If we just correct the X position by adding 8, it will remain in contact with the right wall and look like it was also crushed by the block.

While a collision between a missile and a block isn't difficult to detect, the program must determine which block was hit. Obviously, the block must be in the row directly in line with the pistol. If you look at the diagram below, you will see that the pistol is sixteen scan lines below the top of the man. Since the top left position of the man is at X,Y, then YM = Y+16. The bullet fires from the tip of the gun at XM = X when facing left, and at XM = X + 8 when facing right. The movement routine prevents the pistol from going beyond the top of the first row of blocks. When the pistol is at the top of the first row of blocks (YM = 48) the man is at Y=32. Therefore, the formula ROW =INT((YM -32)/16) determines which row the bullet travels. If we substitute YM = Y+16 the expression simplifies to ROW = INT((Y -16)/16). For example, if the man is at the very top (Y=32) then his pistol is aimed along ROW = INT (32–16)/16 or ROW = 1 as expected. The bullet moves 2 pixels horizontally each frame until it eventually collides with a block. The pairs of blocks are then removed from the side the player faces. Since the player can't move while the bullet is in motion, there is no ambiguity possible. For example, if the player (PLAY=0) were facing left on row 2, and the left block pointer L(2)= 9 , then blocks 2,9 and 2,8 are removed. You need only POKE a 0 into locations SCREEN+OFFSET and SCREEN+OFFSET-1 where OFFSET = 40*ROW + L(ROW). Likewise, if a player



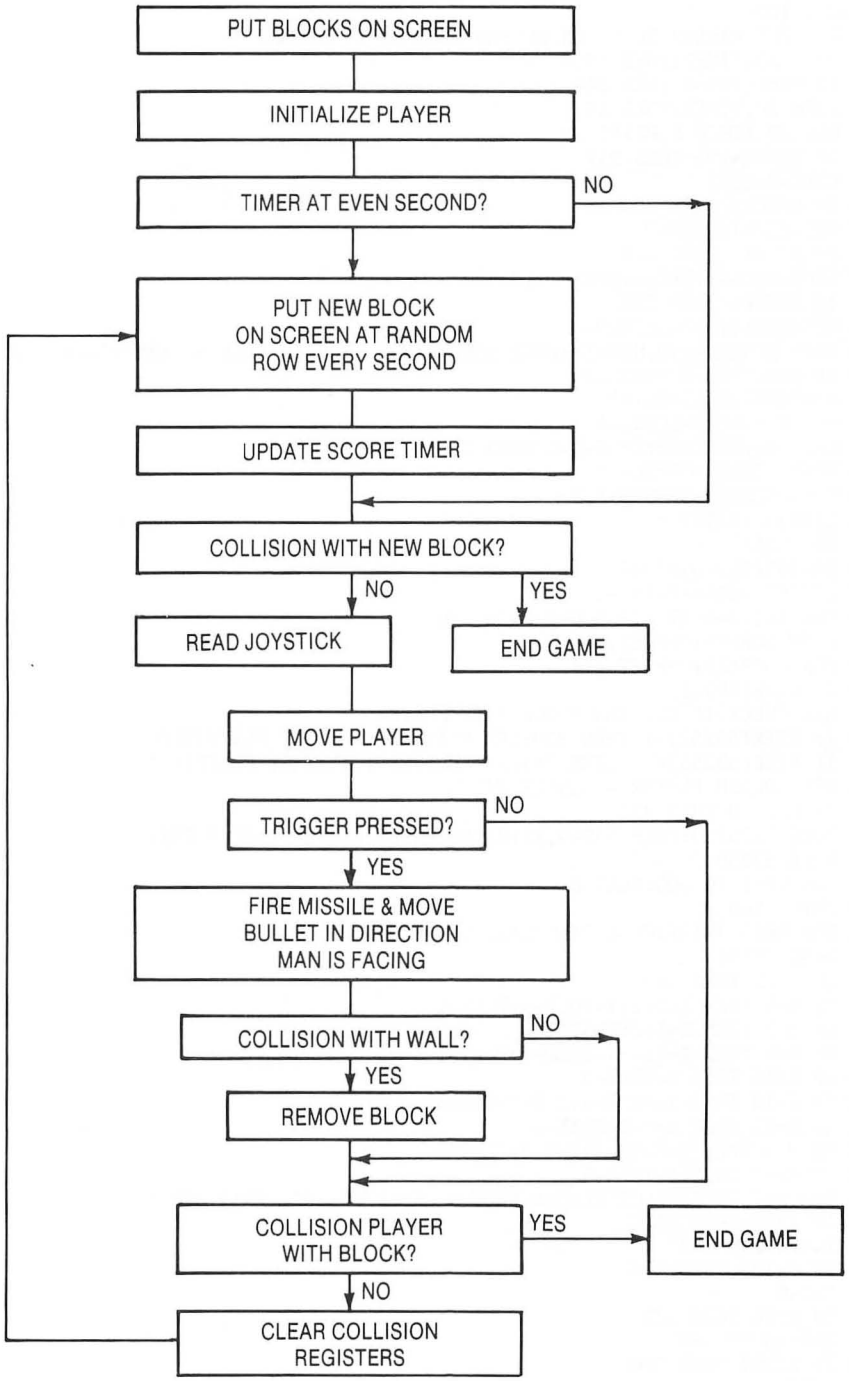
were facing right (PLAY=1), the two blocks that need to be removed are at locations SCREEN+OFFSET and SCREEN+OFFSET +1 where OFFSET = 40*ROW + R(ROW).

The game naturally ends when the player runs out of space. The stopped timer indicates the time elapsed. At this point everything has to be reset for the next game. The blocks on the screen are reset for the next game. The blocks on the screen are reset slowly due to Atari BASIC's naturally slow implementation of the POKE statement. This does give a breather between games. The timer is reset, then the player. A button press is all that you need to begin a new game.

Extra Colors via a Display List Interrupt Subroutine

Some of the advanced programmers might find the display list interrupt subroutine to be of interest. It changes the color value in color register 1 ($2C5) each time ANTIC calls it via a display list interrupt. Since it keeps an internal counter called PLACE for its X register index, it must know when to reset its pointer. It does this by checking the value in the vertical line counter which increments by one for every two scan lines. When it is equal to line 50, which is two scan lines beyond the beginning of row #1, it resets PLACE to 0. It loads PLACE into the X register, indexes into the color table, and stores it in the color register. Now PLACE is incremented by one. The next DLI just loads the next color in the table into the color register. The pushes and pulls on the stack at the beginning and end of our subroutine save the current X register and Accumulator so that they are restored upon return to your program.

SHOOT BLOCKS GAME



```
10 REM - SHOOT BLOCKS GAME - BY JEFFREY STANTON
12 POKE 106,PEEK(106)-12
15 DIM CT(16),L(10),R(10)
20 GOTO 1000
199 REM PUT RANDOM BLOCK ON SCREEN ONCE EVERY SECOND
200 POKE 20,TIMER:POKE 19,0
210 IF PEEK(19)=0 THEN 344
212 POKE 20,TIMER:POKE 19,0
213 REM UP TIMER & PRINT
214 IF TIMER=195 THEN 217
215 HSEC=HSEC+1
216 IF HSEC<2 THEN 220
217 SEC=SEC+1:HSEC=0
220 IF SEC<10 THEN 224
222 SECD=SECD+1:SEC=0:POKE 77,0:REM STOP ATRACT
224 IF SECD<6 THEN 228
226 MIN=MIN+1:SEC=0:SECD=0
228 POKE SCREEN+450,MIN+16:POKE SCREEN+452,SECD+16:POKE SCREEN+453,SEC+16
230 IF RND(0)>0.5 THEN 270
235 RR=INT(RND(0)*10+1)
240 OFFSET=40*RR+L(RR)+1
245 C=L(RR):D=R(RR):IF D-C=1 THEN 230
250 POKE SCREEN+OFFSET,1
260 POKE SCREEN+OFFSET+1,1
265 L(RR)=L(RR)+2
267 GOTO 344
270 RR=INT(RND(0)*10+1)
280 OFFSET=40*RR+R(RR)-1
285 C=L(RR):D=R(RR):IF D-C=1 THEN 270
290 POKE SCREEN+OFFSET,1
300 POKE SCREEN+OFFSET-1,1
305 R(RR)=R(RR)-2
309 REM CHECK IF ANY NEW BLOCK HITS PLAYER
310 IF PEEK(53252)>1 THEN 320:REM PLAYER 0 AGAINST PLAYFIELD
315 IF PEEK(53253)<>2 THEN 344:REM PLAYER 1 AGAINST PLAYFIELD
319 REM SQUASH PLAYER - SINGLE WIDTH
320 IF PLAY=0 THEN 322
321 POKE 53257,0:POKE 53249,X1+8:REM MOVES PLAYER 1 RT 8 UNITS
322 POKE 53256,0
325 FOR DE=1 TO 500:NEXT DE
330 GOTO 1500
343 REM READ JOYSTICK & CALCULATE NEW POSITION
344 Z=STICK(0)
345 IF Z=15 THEN 360
346 IF Z=5 THEN X=X+2:Y=YOLD+4:PLAY=1
347 IF Z=7 THEN X=X+2:PLAY=1
348 IF Z=6 THEN X=X+2:Y=YOLD-4:PLAY=1
349 IF Z=14 THEN Y=YOLD-4
350 IF Z=10 THEN Y=YOLD-4:X=X-2:PLAY=0
351 IF Z=11 THEN X=X-2:PLAY=0
352 IF Z=9 THEN Y=YOLD+4:X=X-2:PLAY=0
353 IF Z=13 THEN Y=YOLD+4
359 REM SET UP CORRECT PLAYER PLOT & CHECK DOESN'T EXIT SCREEN
360 IF Y>32 THEN 364
362 Y=32:GOTO 368
364 IF Y<191 THEN 368
366 Y=190
368 IF X>46 THEN 372
370 X=46:GOTO 380
372 IF X<207 THEN 380
374 X=206
```

```
380 IF PLAY=1 THEN XO=10:X1=X:GOTO 384
382 XO=X:X1=10
384 A=USR(1536,0,30,YOLD,Y,XO)
386 A=USR(1536,1,30,YOLD,Y,X1)
390 YOLD=Y
392 REM CHECK IF PLAYER HITS ANY BLOCK AFTER MOVE
394 IF PEEK(53252)>1 THEN 398:REM PLAYER 0 AGAINST PLAYFIELD 1
396 IF PEEK(53253)<>2 THEN 410:REM PLAYER 1 AGAINST PLAYFIELD 1
397 REM SQUASH PLAYER - SINGLE WIDTH
398 IF PLAY=0 THEN 322
399 POKE 53257,0:POKE 53249,X1+8:REM MOVES PLAYER 1 RT 8 UNITS
400 POKE 53256,0
402 FOR DE=1 TO 500:NEXT DE
405 GOTO 1500
409 REM FIRE MISSILE
410 IF STRIG(0)<>0 THEN 600
411 FOR L1=10 TO 4 STEP -0.25
412 SOUND 0,10,0,L1:NEXT L1
413 POKE 77,0
415 ROW=INT((Y-16)/16)
420 IF PLAY=0 THEN 470
430 XM=X+10:YM=Y+16
435 FOR J=1 TO 100
440 XM=XM+2
445 A=USR(1536,4,2,YMOLD,YM,XM)
446 YMOLD=YM
448 IF XM>206 THEN 597
450 IF PEEK(53248)>0 THEN 500
455 NEXT J:GOTO 597
470 XM=X:YM=Y+16
475 FOR J=1 TO 100
480 XM=XM-2
485 A=USR(1536,4,2,YMOLD,YM,XM)
486 YMOLD=YM
488 IF XM<47 THEN 597
490 IF PEEK(53248)>0 THEN 500
495 NEXT J:GOTO 596
499 REM TO REMOVE SHOT BLOCK
500 ROW=INT((Y-16)/16)
502 FOR L1=15 TO 0 STEP -0.5
503 SOUND 0,20,2,L1:NEXT L1
505 POKE 53278,0:REM CLEAR COLLISION REGISTER
510 IF PLAY=1 THEN 560
520 C=L(ROW)
525 IF C=-1 THEN 600
530 OFFSET=40*ROW+C
540 POKE SCREEN+OFFSET,0
550 POKE SCREEN+OFFSET-1,0
554 L(ROW)=L(ROW)-2
555 POKE 53278,0:YM=5
556 A=USR(1536,4,2,YMOLD,YM,10):REM REMOVE MISSILE TO EDGE
557 YMOLD=YM:GOTO 600
560 C=R(ROW)
565 IF C=40 THEN 600
570 OFFSET=40*ROW+C
580 POKE SCREEN+OFFSET,0
590 POKE SCREEN+OFFSET+1,0
595 R(ROW)=R(ROW)+2
596 POKE 53278,0:YM=5
597 A=USR(1536,4,2,YMOLD,YM,10):REM REMOVE MISSILE TO EDGE
598 YMOLD=YM
```

```
600 GOTO 210
1000 CB=PEEK(106):CHRSET=CB*256
1005 PM=PEEK(106)+4:PMBASE=PM*256
1010 GRAPHICS 1+16
1020 DLIST=PEEK(560)+PEEK(561)*256
1030 REM CHANGE DISPLAY LIST TO ANTIC 5
1040 FOR I=0 TO 19
1050 READ A:POKE DLIST+I,A:NEXT I
1052 REM POKE IN PM ROUTINE
1053 FOR I=0 TO 150:READ X:POKE 1536+I,X:NEXT I
1055 REM POKE IN PM CLEAR ROUTINE
1056 FOR I=0 TO 25:READ X:POKE 1696+I,X:NEXT I
1059 REM POKE IN HOR DISPLAY LIST INTERUPT ROUTINE
1060 FOR I=0 TO 41:READ X:POKE 1728+I,X:NEXT I
1070 FOR L1=0 TO 511:POKE CHRSET+L1,PEEK(57344+L1):NEXT L1
1075 REM REWRITE 1ST CHARACTER; NOTE 1ST CHARACTER IS NOT THE 0TH CHARACTER
1080 FOR C2=CHRSET+8 TO CHRSET+15:POKE C2,170:NEXT C2
1139 REM SET UP SCREEN
1140 SCREEN=PEEK(88)+PEEK(89)*256
1150 FOR R=1 TO 10:REM 0TH ROW EMPTY
1160 FOR C=0 TO 11
1170 OFFSET=40*R+C
1180 POKE SCREEN+OFFSET,1
1190 NEXT C
1200 FOR C=28 TO 39
1210 OFFSET=40*R+C
1220 POKE SCREEN+OFFSET,1
1230 NEXT C:NEXT R
1232 REM READ TIME DATA LINE
1234 FOR I=0 TO 13:READ A:POKE SCREEN+440+I,A:NEXT I
1236 TIMER=195:HSEC=0
1240 FOR I=1 TO 10
1250 L(I)=11:R(I)=28:NEXT I
1260 POKE 756,CB
1265 POKE 512,192:POKE 513,6:REM VECTOR TO CODE LOCATION FOR DLI
1266 POKE 54286,192:REM ENABLE DISPLAY LIST INTERUPT
1270 REM ACTIVATE P/M
1280 PLAY=1
1290 SETCOLOR 2,0,0
1300 POKE 559,62:REM SET DMACTL -SINGLE LINE
1302 POKE 53277,3:REM SET GRACTL -PLAYERS&MISSILES
1310 A=USR(1696,PMBASE)
1320 POKE 53256,1:POKE 53257,1:REM PLAYERS DOUBLE WIDTH
1325 POKE 53260,1:REM MISSILE 0 DOUBLE WIDTH
1330 POKE 704,216:POKE 705,216:REM PLAYER COLORS
1335 POKE 54279,PM:REM TELL ANTIC PMBASE
1340 POKE 1686,PM:REM POKE HI BYTE PMBASE INTO SUBROUTINE
1350 Y=50:YOLD=50:X0=10:X1=125:X=125
1359 REM READ PM DATA INTO STORAGE AREA
1360 FOR I=0 TO 29
1370 READ VAL:POKE PMBASE+I,VAL:NEXT I
1380 FOR I=0 TO 29
1390 READ VAL:POKE PMBASE+64+I,VAL:NEXT I
1400 REM SELECT KEY CHANGES LEVEL & START BEGINS GAME
1401 REM TIMER=195 SLOW 1 SEC/BLOCK
1402 REM TIMER=225 FAST 1/2 SEC/BLOCK
1403 REM * MARKS DIFFICULT LEVEL
1405 IF PEEK(53279)<>5 THEN 1420
1410 IF TIMER=195 THEN TIMER=225:POKE SCREEN+440,10:GOTO 1420
1415 IF TIMER=225 THEN TIMER=195:POKE SCREEN+440,0
```

```
1420 IF PEEK(53279)=6 THEN 1450
1425 FOR DE=1 TO 10:NEXT DE:GOTO 1405
1450 GOTO 200
1499 REM RESET SCREEN FOR REPLAY
1500 FOR R=1 TO 10
1510 FOR C=0 TO 11
1520 OFFSET=40*R+C
1530 POKE SCREEN+OFFSET,1
1540 NEXT C
1550 FOR C=12 TO 27
1560 OFFSET=40*R+C
1570 POKE SCREEN+OFFSET,0
1580 NEXT C
1590 FOR C=28 TO 39
1600 OFFSET=40*R+C
1610 POKE SCREEN+OFFSET,1
1620 NEXT C:NEXT R
1630 FOR I=1 TO 10
1640 L(I)=11:R(I)=28:NEXT I
1650 REM RESET PLAYER TO DOUBLE WIDTH AT CENTER
1670 POKE 53256,1:POKE 53257,1:REM DOUBLE WIDTH
1680 XO=10:X1=125:Y=50:PLAY=1:X=125
1685 HSEC=0:SEC=0:SECD=0:MIN=0
1687 POKE SCREEN+450,MIN+16:POKE SCREEN+452,SECD+16:POKE SCREEN+453,SEC+16
1690 A=USR(1536,0,30,YOLD,Y,XO)
1700 A=USR(1536,1,30,YOLD,Y,X1)
1705 POKE 53278,0:REM CLEAR COLLISION
1710 GOTO 200
1999 REM DISPLAY LIST DATA
2000 DATA 112,112,112,69,128,145,133,133,133,133,133,133,133,133,133,133,7,65,96,145
10000 REM PLAYER SUBROUTINE DATA
10005 DATA 104,162,0,104,104,157,145,6,232,224,5,208,246,173,149,6,174
10006 DATA 145,6,157,0,208,173,150,6
10010 DATA 133,213,24,105,4,133,207,173,145,6,201,4,176,47,170,189,133
10015 DATA 6,133,212,202,48,4,230,207
10020 DATA 208,249,173,147,6,133,206,169,0,168,145,206,200,204,146,6
10025 DATA 144,248,173,148,6,133,206,160,0
10030 DATA 177,212,145,206,200,204,146,6,144,246,96,56,233,4,170,198
10035 DATA 207,173,147,6,133,206,160,0,177
10040 DATA 206,61,137,6,145,206,200,204,146,6,144,243,173,148,6,133,206
10045 DATA 160,0,177,206,29,141,6,145
10050 DATA 206,200,204,146,6,144,243,96,0,64,128,192,252,243,207,63
10055 DATA 3,12,48,192,0,0,0,0,0,0
10069 REM CLEAR PM AREA ROUTINE DATA
10070 DATA 104,104,133,213,104,133,212,162,0,160,0,169,0,145,212,200
10075 DATA 208,251,230,213,232,224,8,144,240,96
10079 REM HOR DISPLAY LIST ROUTINE DATA
10080 DATA 72,138,72,173,11,212,201,32,176,5,169,0,141,223
10085 DATA 6,174,223,6,189,224,6,141,23,208,238
10090 DATA 223,6,104,170,104,64,0
10094 REM COLOR DATA IN ROUTINE
10095 DATA 68,148,20,242,36,180,116,52,164,100
10099 REM "TIME 0:00" DATA
10100 DATA 0,0,0,0,0,52,41,45,37,0,16,26,16,16
10105 REM PLAYER DATA
10110 DATA 24,24,126,126,24,24,56,56,24,24,8,8,28,28,28
10115 DATA 28,252,252,60,60,28,28,8,8,8,8,8,8,56,56
10120 DATA 24,24,126,126,24,24,28,28,24,24,16,16,56,56,56
10125 DATA 56,63,63,60,60,56,56,16,16,16,16,16,16,28,28
```

Space War Game

Space War, the first game with a fully steerable spaceship, was developed at MIT. While most of the newer computer owners won't remember this game, practically everyone is familiar with *Asteroids.* Most versions of this game have a fully steerable spaceship that can be thrust in the direction that it is headed. Although some versions invoke an automatic deceleration mode, some *Asteroid* games require the player to turn his ship around so that it thrusts in the opposite direction to slow down.

Dynamics of Motion with Acceleration

We demonstrated earlier in this chapter that objects move in the direction of their velocity vector. An object's new position is its old position plus its change in position due to velocity.

Using the Atari's screen coordinate system for the example above, VY is negative and VX is positive. Therefore,

$$X = X + VX$$
$$Y = Y + (-VY)$$

While the velocity vector may remain constant for many animation cycles, so that a ship will continue to move in the same direction, sooner or later a new velocity vector will be input to change the object's course. This new velocity is the vector sum of the old velocity vector and the new velocity vector.

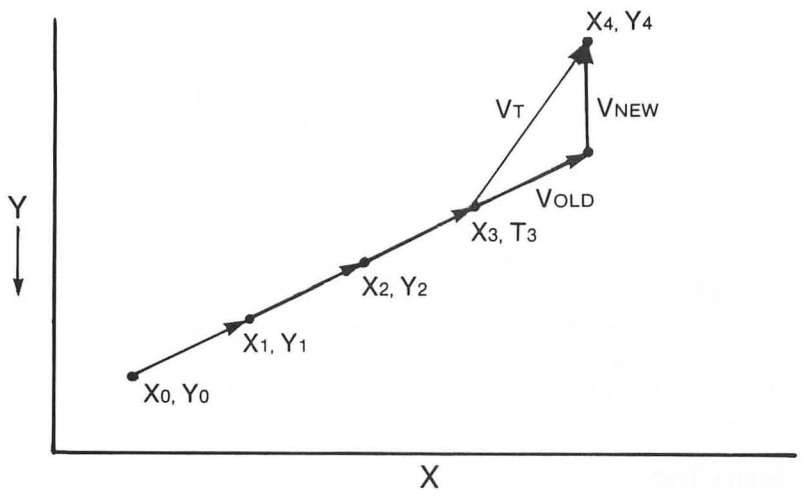
Those readers who have taken Physics will recall that the velocity of a body in motion changes due to external forces on it while it is in motion. In spaceships, that force is thrust. Thrust causes an acceleration of the object's mass as shown in the equation.

$$F = m * a = m * \Delta V$$

When thrust is applied to a spaceship, it accelerates. If a ship is light and has a big engine with considerable thrust, it will accelerate quickly. But if it is heavy, it will accelerate much slower. Acceleration is essentially caused by a change in the object's velocity if you ignore the object's mass.

Unless you are doing an actual simulation, in which the values of thrust or force and an object's mass are important, only acceleration values need to be considered. Suitable values for arcade games are small and scaled, so that objects don't move fast relative to their size, or fly off the screen in the blink of an eye.

If we consider a spaceship that is in motion for three frames, then thrust only during the fourth frame, it will change direction depending on the vector sum of its old and new velocity vectors. This is illustrated below. The applied thrust is straight



upwards, so that VX = 0 and VY = –2. The ship's new velocity vectors for each direction are calculated as follows:

$$VX = VX(old) + \Delta VX = 2 + 0 = 2$$
$$VY = VY(old) + \Delta VY = -1+(-2) = -3$$

Likewise, the ship's new position is equal to its old position plus its change in position due to velocity for that animation frame. This breaks down into components for the X and Y directions.

$$X = X(old) + VX$$
$$Y = Y(old) + VY$$

The ship's new velocity vector causes it to move two units in the X direction and three in the negative Y direction during each frame until a new thrust vector is applied. The resultant position can be summarized in the table below.

FRAME	X	Y	ΔVX	ΔVY	
0	10	100	2	-1	
1	12	99	2	-1	
2	14	98	2	-1	
3	16	97	2	-3	Thrust applied here
4	18	94	2	-3	
5	20	91	2	-3	

Now, when the acceleration on an object is sustained over many animation frames, the increase in velocity is cumulative. For example, if thrust were applied to a stationary object in the positive X direction with a force of 1 unit/frame, the new VX would increase from zero by units of one for each animation frame.

	CYCLE	VX	X		CYCLE	VY	Y
	0	0	0		0	0	0
	1	1	1		1	2	2
VX=1	2	2	3	Similarly VY=2	2	4	6
	3	3	6		3	6	12
	4	4	10		4	8	20

Our example makes it clear that if you accelerate for too many animation frames, the spaceship will be moving too fast. A limit should be set on the velocity vector for both directions. Just what the limit should be depends on the effect you wish to achieve. Obviously, if your animation rate is that of VBLANK, 60 frames per second, a ship velocity of 5 units/cycle will race across the screen in one-half second.

Joystick Routine

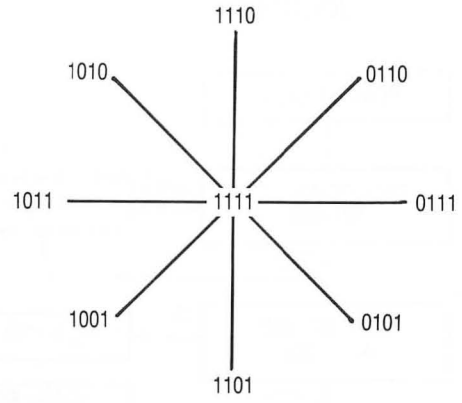
A joystick will control the ship's direction in our game. Pushing to the left or right rotates the ship to one of eight directions. Pushing up thrusts the ship. The joystick is read in the VBLANK routine every 1/60th of a second. While a Machine language subroutine is more than equal to the task of keeping up with the update, its speed creates a small problem. If a player turns his ship by pushing left or right on his joystick, the ship will spin rapidly, making 1/8 of a turn every 1/60 of a second. That translates to 7 and 1/2 turns per second, a speed that is obviously uncontrollable. The joystick must be read less often, say every sixth VBLANK cycle if the ship is to turn slowly enough to steer.

Machine language joystick routines are inherently simpler because only four bit positions for up, down, left and right, need to be tested. Diagonals are usually ignored because they nearly always represent combinations of commands. For example, a diagonal up and right command in our game means thrust while turning right. The joystick read subroutine will detect the bit pattern twice, once when testing the up bit to determine if it should reset the velocity, and a second time when testing the right bit to determine if it should turn the ship. The bit pattern is as follows.

X	X	X	X	RIGHT	LEFT	DOWN	UP
128	64	32	16	8	4	2	1

All of the bits are normally set when the joystick is centered or in the neutral position (1 1 1 1). When the joystick is pushed in any direction its particular bit position is turned off, as illustrated in the diagram below.

To test a particular bit position you only need to AND it with that bit. For example, to test if the left bit is turned off:

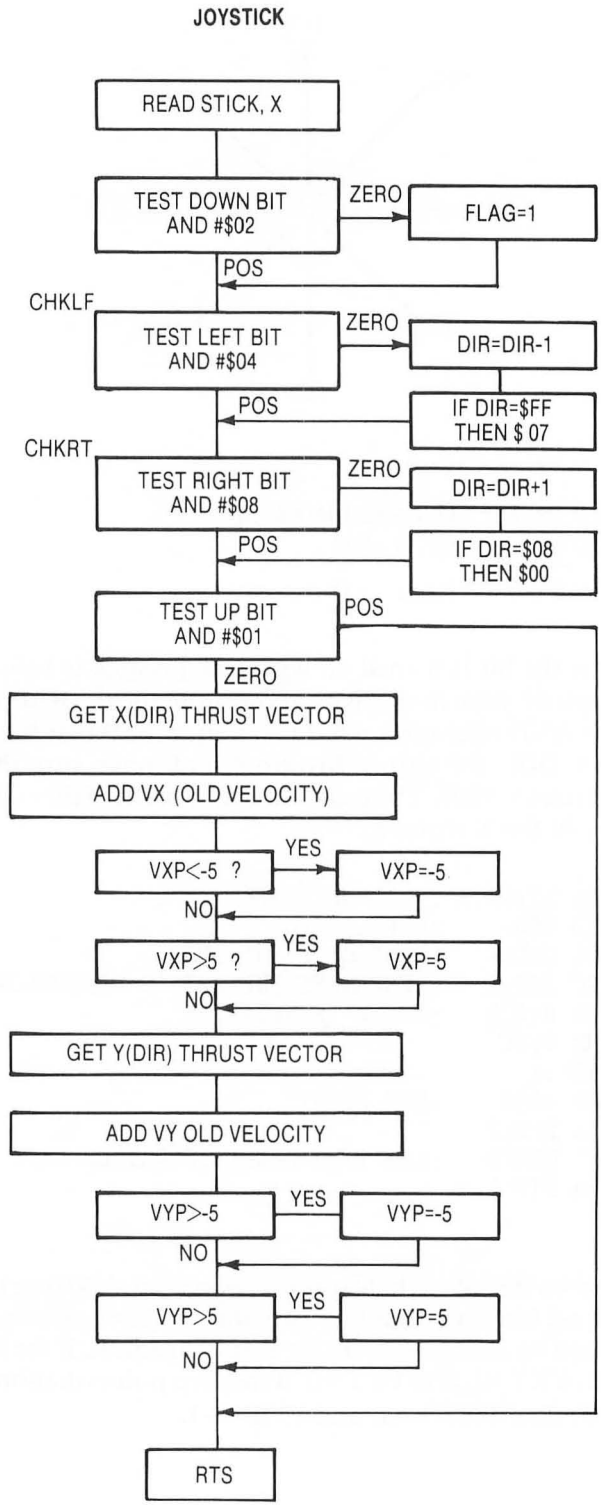



```
0 0 0 0 1 0 1 0   Diagonal left & up
0 0 0 0 0 1 0 0   AND #$04
-----------------
0 0 0 0 0 0 0 0   Result = 0
```

The result is zero if the bit is turned off when the joystick is being pushed in that direction. If the joystick were in the neutral position, the third bit would be on, and the result after the AND operation would be non-zero. When the result is zero we want to decrement DIR, the ship's direction, and make sure that if it becomes negative ($FF) it is reset to #$07. The code is as follows. All of the code is indexed with the player number in the X register.

```
CHKLF    LDA STICK,X ;READ JOYSTICK
         AND #$04    ;LEFT BIT
         BNE CHKLF   ;BRANCH IF NOT ZERO
         DEC DIR,X   ;DECREMENT THE SHIP'S DIRECTION
         LDA DIR,X   ;CHECK IF NEGATIVE
         CMP #$FF
         BNE .1
         LDA #$07    ;SET DIR=7
         STA DIR,X
.1       JMP CHKFD   ;CAN'T BE RIGHT IF PUSHED LEFT
CHKRT    LDA STICK,X
           .    .
```

Pushing forward on the joystick thrusts the spaceship. ANDing the joystick value with #$01 tests the up bit. If the result is zero, the joystick has been pushed up. The acceleration or thrust vector depends on the ship's direction. If the ship points to the right, DIR = 2, then VXT = 1, and VYT = 0. If the ship points diagonally upward and to the left, DIR = 7, then VXT = -1, and VYT = -1.


JOYSTICK
READ STICK, X
TEST DOWN BIT AND #$02
ZERO
FLAG=1
POS
CHKLF
TEST LEFT BIT AND #$04
ZERO
DIR=DIR-1
POS
IF DIR=$FF THEN $ 07
CHKRT
TEST RIGHT BIT AND #$08
ZERO
DIR=DIR+1
POS
IF DIR=$08 THEN $00
TEST UP BIT AND #$01
POS
ZERO
GET X(DIR) THRUST VECTOR
ADD VX (OLD VELOCITY)
VXP<-5 ?
YES
VXP=-5
NO
VXP>5 ?
YES
VXP=5
NO
GET Y(DIR) THRUST VECTOR
ADD VY OLD VELOCITY
VYP>-5
YES
VYP=-5
NO
VYP>5
YES
VYP=5
NO
RTS

Ship's Direction and Velocity Vectors

Note that many of our ship's directions produce negative velocity values, while others produce positive values. Separate routines are required for adding and subtracting in Machine language. BASIC, however, just adds a negative number (X = 5+ (−1)). That's the clue. Adding a negative number is exactly the same as adding a positive number in Machine language. The difference is that negative numbers, like −1, are represented by the two's complement which for −1 is $FF. There is a limit for signed numbers of + or −127, because the BMI instruction tests the carry bit and considers the value if set. With the simplification of our thrust vector addition problem, we can construct a table of velocity vectors for each DIR value.

TRUST VECTOR

DIR	0	1	2	3	4	5	6	7
VTX	00	01	01	01	00	FF	FF	FF
VTY	FF	FF	00	01	01	01	00	FF

The equations for the ship's two velocity vector components are as follows:

VXP = VX(old) + VTX(DIR)
VYP = VY(old) + VTY(DIR)

A speed brake can be incorporated into the algorithm to prevent the velocity from exceeding a preset value. This would be analogous to wind resistance on a fast-moving automobile. It prevents a vehicle's speed from increasing infinitely. I choose a maximum velocity of 5 units per frame. It is based on keeping the animation smooth and the speed in bounds. The code for the X direction is as follows:

```
        LDA DIR,X    ;GET PLAYER DIR
        TAY
        CLC
        LDA VTX,Y    ;GET X(DIR) THRUST VECTOR
        ADC VX       ;ADD OLD VELOCITY VECTOR
        CMP #$FA     ;IS IT -6?
        BNE .5
        LDA #$FB     ;CLIP TO -5
.5      CMP #$06
        BNE .6
        LDA #$05     ;CLIP TO 5
.6      STA VX
        STA VXP,X    ;STORE NEW SHIP VELOCITY IN X DIRECTION
```

Addressing the Correct Shape Table

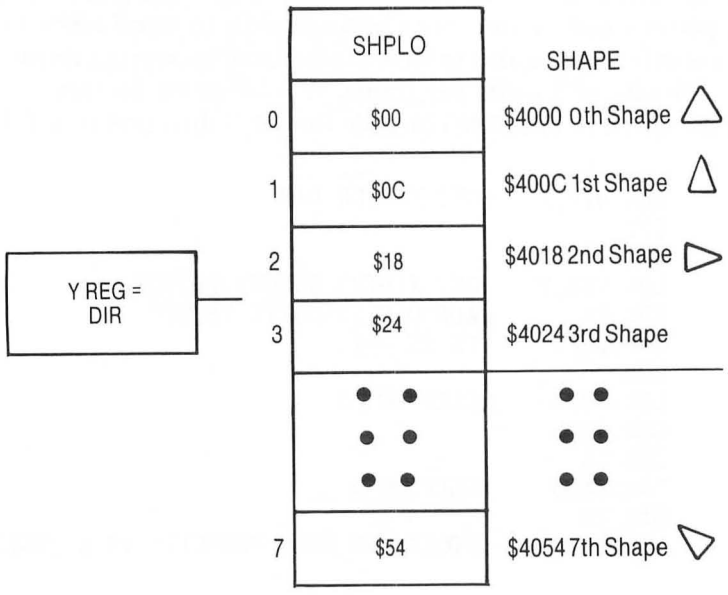
Now that we can control our ship in eight directions, we need shape tables for each of these directions. That means eight separate shapes, each twelve bytes long. The plot subroutine that places the shape in the player-missile memory area is virtually the same as that used in our player-missile subroutine discussed earlier in the chapter.

The zero page pointers to our shape and to its eventual storage location in player-missile graphics memory are set up in a subroutine called PLOTSET0 for player #0 and PLOTSET1 for player #1. Since our drawing routine takes the direction into consideration in order to obtain the correctly rotated shape from our shape table, we can find the correct low byte of the shape by the following formula:

SHPL = SHPLO(DIR)

The shape number DIR, which is also our direction, is placed in the Y register so that we can find the low byte pointer to our shape stored in a table called SHPLO. Each of the values in that table are twelve bytes apart starting at #$00. The high byte is constant for all shapes.

```
LDY DIR      ;VALUE FOR DIRECTION OF ROTATED SHAPE
LDA SHPLO,Y  ;AS INDEX TO PROPER LOW BYTE OF SHAPE
STA SHPL     ;STORE LOW BYTE POINTER IN ZERO PAGE
LDA /SHAPE0  ;HIGH BYTE
STA SHPH     ;STORE HIGH BYTE IN ZERO PAGE
```



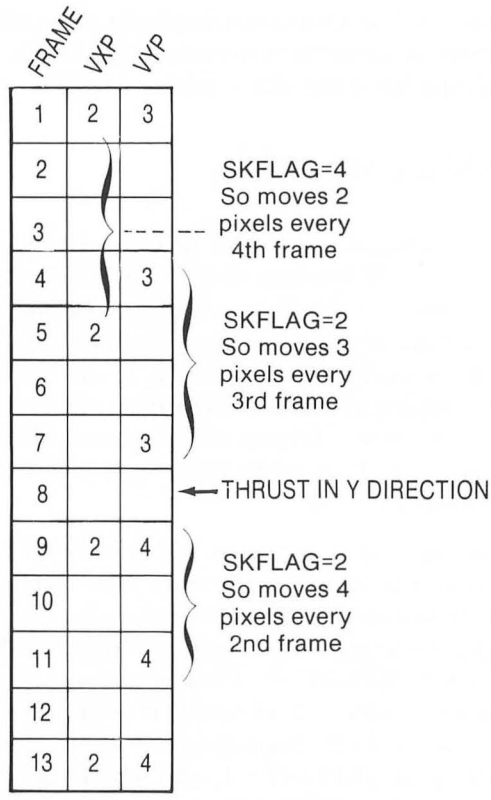
If the ship were turned so that it was pointing right, then DIR = 2 and SHPLO(2) = $18. This low byte of the shape table is stored as SHPL. The drawing routine will now plot the second shape from our shape table.

Smoothing the Ship's Movement

Recall that when we updated our ship's position in our last BASIC example, movement wasn't very smooth because of the slow animation frame rate. Potentially, we face a very similar problem here because the ship's velocity vector that controls its next position is only updated every sixth frame. If the ship's position is updated in the same loop as the ship's velocity, it will appear to have very jerky movement, sometimes moving as much as five pixels per frame. It would be better to move the ship in smaller increments more often, even as fast as the scan rate of sixty times per second. Of course, at slower velocities the ship would have to be moved less often. The trick is to control the rate.

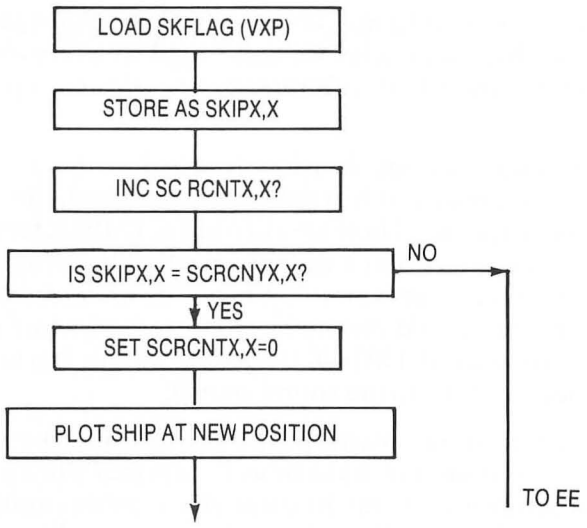
If the ship were moving in a direction at full speed, VX = 5, we would want to update the position every frame, but if the ship were moving slowly, VX = 1, we would want to update the screen every fifth or sixth cycle. This means that we should skip plotting the ship at its new position until a number of frames has passed. We could use a counter called SKFLAG to check when we should plot. Naturally, it would be different for each velocity and could be stored in a lookup table for easy access. Its values would nearly be the reciprocal of the velocity. At VXP = 5 we would want to replot each frame so SKFLAG = 1, and when VXP = 1 we would want to replot every fifth or sixth cycle or SKFLAG = 6. The relationship between the ship's movement and its actual velocity VXP isn't exactly linear. If you look in the table you will notice that it is fairly linear at slow speeds, but jumps from twenty pixels every sixty frames at VXP = 3, to thirty pixels every sixty frames at VXP = 4, to a whopping sixty pixels every sixty frames at VXP = 5. Fortunately, this is only a game, and the discrepancy is not noticeable.

INDEX	0	1	2	3	4	5	6	7	8	9	10
SKFLAG	01	02	03	04	06	06	06	04	03	02	01
VELOCITY VXP or VYP	—5 F(FB)	—4 (FC)	—3 (FD)	—2 (FE)	—1 (FF)	0	1	2	3	4	5
MOVEMENT IN 60 FRAMES	60	20	30	15	10	0	10	15	20	30	60



The code that determines when the ship is to be moved and replotted for each direction is quite simple. The number of frames to skip, SKFLAG, is obtained from our table based on the ship's current velocity and direction. Five has been added to the velocity so that negative VXP, and VYP values can be indexed, too, using the Y register. Counters for each axis, SCRCNTX and SCRCNTY, that keep track of the number of frames elapsed since the last screen update, are incremented. They are then compared against the values from our table. When it matches, that counter is reset to zero, and the player's position for that axis is updated and plotted. The code for the X axis is listed below.

```
UPDATEX  CLC             ;LOAD PLAYER'S HORIZ. VELOCITY
         LDA VXP,X       ;SO NEG #'S APPEAR IN TABLE TOO
         TAY             ;USE AS INDEX
         LDA SKFLAG,X    ;GET VALUE FROM TABLE
         INC SCRCNTX,X   ;INC. COUNTER FROM LAST UPDATE
         CMP SCRCNTX,X   ;AT UPDATE TIME?
         BGE .1
         JMP EE          ;NO! DON'T UPDATE
.1       LDA #$00        ;RESET COUNTER
         STA SCRCNTX,X
         LDA VXP,X       ;BEGIN UPDATING PLAYER POSITION
          .   .
```



Each time the player's position is updated in either axis, the player moves one pixel position in the proper direction. If the velocity is negative, the player moves either to the left or up depending on which axis is being updated. If the velocity is positive, the player moves down or to the right. Remember we decided to move the ship only a single pixel at a time because moving the player in finer increments at higher frame rates produces smoother animation than discontinuous jumps at slow frame rates. The ship's overall velocity is the same, but the ship doesn't strobe as it moves.

The screen has a wraparound feature in this game. This means that when a ship reaches the right side of the screen and exits, it reappears on the left side with its velocity and direction the same. This is true in the vertical direction, too. A ship leaving the top of the screen will reappear at the bottom. All that is needed is a simple check to determine if the ship has reached the screen boundary. If it has, its position is reset to the opposite screen boundary. Since the boundaries of the playfield screen don't quite reach the edges of the television set, we extended the screen boundary coordinates by eight pixels in all directions. The ship's vanishing point should be nearly at the edge. If a ship is traveling to the right, it will vanish at X = 216 ($D8) and reappear at X = 40 ($28). Don't confuse player-missile coordinates with the screen coordinates used in PLOT and DRAWTO statements. The top left corner of the playfield in player-missile coordinates is X = 48, Y = 32. Coordinate 0,0 is offscreen.

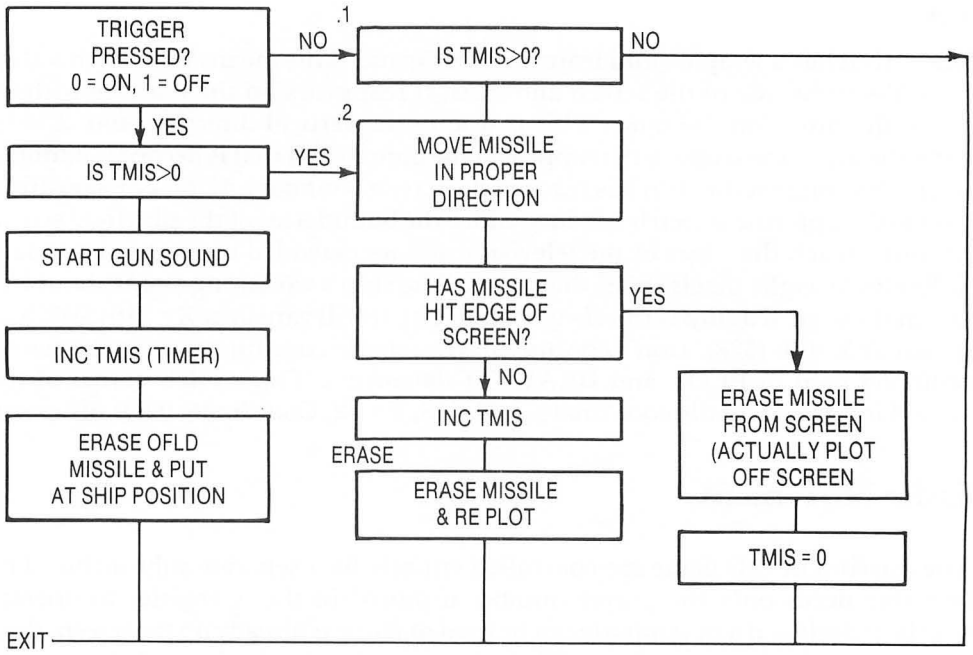
Missile Movement

The missiles in this game are controlled entirely by a separate subroutine. The subroutine needs only the player number inputted in the X register to operate correctly. It decides if a new missile can be fired or if one is already on the screen, then moves the missiles appropriately. There is no need to read a joystick for missile

direction as in our earlier game, because the missile fires and moves in the direction, DIR, that our ship faces. Also we don't need to worry about updating the ship's position while in the missile subroutine, since the main program loop takes care of this.

The subroutine's initial decision is whether the joystick trigger STRIG0,X ($284,X) is being pressed. When the button is pressed, the subroutine returns a zero, and, if untouched, a one. There are also timers TMIS,X for each missile, to count the number of screen cycles that a missile travels. The actual values are unimportant, except whether they are zero or not. When TMIS,X is greater than zero, the missile is already on the screen and merely has to be moved while checking for screen edge boundaries. However, if TMIS,X is zero, the missile has to be placed initially at the ship's position, and the firing sound started.

Since we only have one missile per player, we can't have rapid fire missiles or more than one missile on the screen at a time. Games that allow a train of five or six missile tracks on the screen at one time are using character set animation or bit mapping, not missiles. We could have allowed the player to reset his missile track each time he pressed the trigger, but instead we decided that you couldn't refire until the missile reached the screen's edge or hit its target. Since many players might hold the trigger down indefinitely we had to test in both branches for whether the missile was already on the screen. If we had omitted this test, the missile track would reset each time a player pressed the trigger. If you wish to change the game to this mode of firing, just remove the test.



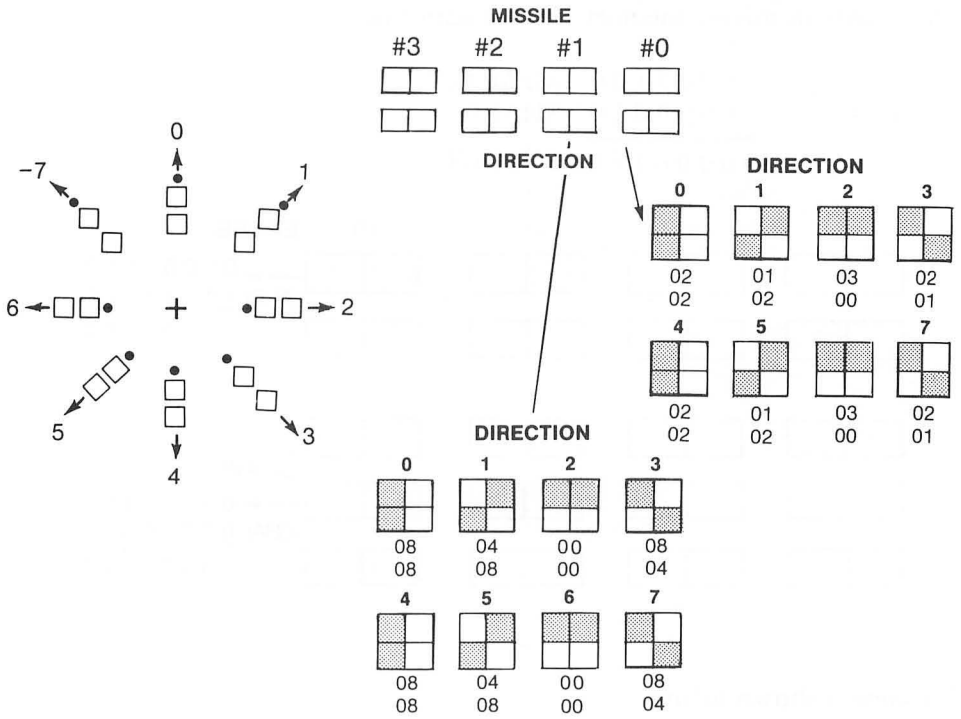
Once a missile is launched, it must continue to move in the initial direction, regardless of a change in course of the mother ship. We accomplish this by storing its initial direction at launch DIR,X as DIRM0,X. This direction is used to look up the missile velocity vectors VTX and VTY from tables. The position is updated by the following formula:

$$XMIS(new) = XMIS(old)+VTX(DIRM0)$$
$$YMIS(new) = YMIS(old)+VTY(DIRM0)$$

The code is as follows:

```
.2      LDY DIRM0,X ;LOAD Y REG. WITH MISSILE'S DIRECTION
        LDA VTX,Y   ;LOOK UP X DIRECTION VELOCITY VECTOR
        ASL         ;DOUBLE VELOCITY VECTOR
        CLC
        ADC XMISO,X ;ADD MISSILE'S OLD X POSITION
        STA XMISO,X ;STORE NEW X POSITION
        LDA VTY,Y   ;LOOK UP Y DIRECTION VELOCITY VECTOR
        ASL
        CLC
        ADC YMISO,X ;ADD MISSILE'S OLD Y POSITION
        STA XMISO,X ;STORE NEW Y POSITION
```

These missile shapes are not the square 2 by 2 blocks of our earlier example. They are two pixels set vertically, horizontally, or diagonally in the direction of travel. A chart of their shapes follows:



First, notice that each missile has a different set of numerical values. Missile shape #0 uses the first two bits in the byte, and missile #1 uses the third and fourth bit positions. Therefore, there is a shape table for each missile. The shapes are stored in two-byte pairs for the two scan line high shapes. There is also an index to the low byte to each of these shapes. They are in two tables called MISLO and MISLO1 for each missile set respectively.

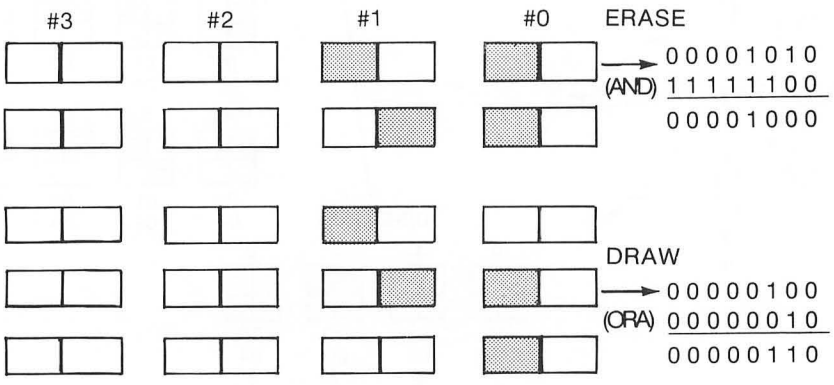
The missile setup subroutine sets up the zero page pointers to the correct shape and to the storage location in player-missile memory. This area is just below the players or .75K from the start of player-missile memory. In addition, the proper mask for that player is stored as MASK.

Plotting the missile requires erasing the old missile first by ANDing the byte containing all four missiles with the proper mask. This step erases only the desired missile because the mask contains zeros in the missile bits to be erased and ones elsewhere. We then plot the missile on the screen by ORAing with a byte in the missile memory that may or may not contain other missiles. For example, if we wish to erase missile #1 in a byte that contains missile #0, we AND it with the mask $FC.

```
        00001010  MEMORY
AND     11111000  MASK
        --------
        00001000  RESULT
```

If we move missile #1 to a scan line that contains missile #3 then we ORA its shape with the byte in missile memory for that scan line.

```
        00000100  MEMORY
ORA     00000010  SHAPE
        --------
        00000110  RESULT
```



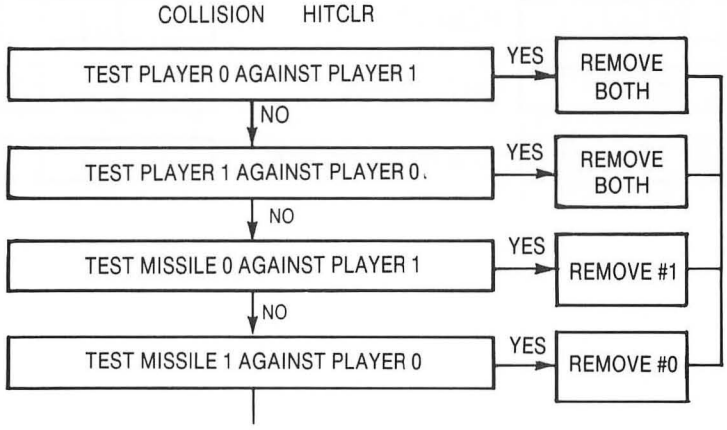
The code is shown below:

```
MPLOT   LDY #$00
.1      LDA (SHPMOL),Y;LOAD OLD SHAPE
        AND MASK      ;MASK WITH PROPER MISSILE BEING MOVED
        STA (SHPMOL),Y;STORING IT DOESN'T ERASE OTHER MISSILES
        INY           ;NEXT BYTE
        CPY #$02      ;MISSILE 2 BYTES HIGH
        BLT .1
        LDY #$00
.2      LDA (SHPL),Y  ;GET BYTE FROM CORRECT SHAPE TABLE
        ORA (SHPML),Y ;ORA AGAINST OTHER MISSILES ON SCAN LINE
        STA (SHPML),Y ;PUT IN MISSILE AREA
        INY           ;NEXT BYTE
        CPY #$02
        BLT .2
```

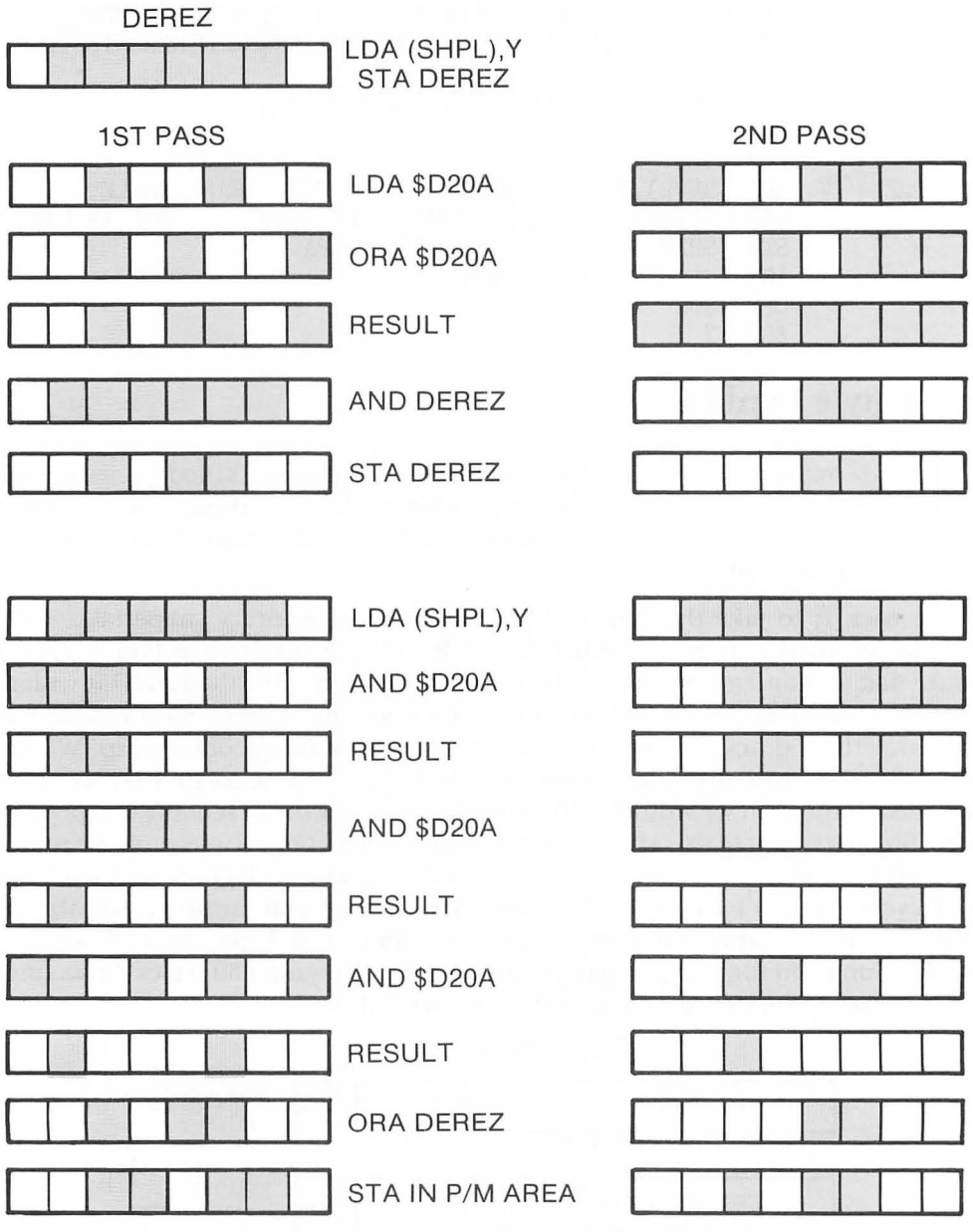
Derez Style Explosion

There are many types of explosions suitable for destroying a killed spaceship. One of the more effective methods is to de-rez the object. This technique makes the object appear to slowly disintegrate by randomly flickering the individual pixels until most of the pixels vanish.

The trick is to take the ship and store it into a temporary shape table called DEREZ. We load a random number, then ORA it with another random number to insure that we don't get a number with a few of bits "on". We then AND it with the shape in DEREZ and store it there. If we then AND the original ship's shape with this value three times, we get a significantly degraded image of the ship. We then ORA this with our degraded temporary shape in DEREZ so that we get a less degraded shape than we would get if we performed only the first of the two previous steps. Since we wanted this effect to last at least 30 cycles or half a second, the routine was largely experimentally determined. The image begins to degrade slightly during each cycle but isn't quite steady. Any byte's image can improve slightly but randomly in any frame, but the overall effect is continued degradation. The loop is 48 cycles long, but the image usually vanishes completely after 30 cycles. An example of two separate passes for a single byte is shown below.



DEREZ EXPLOSION

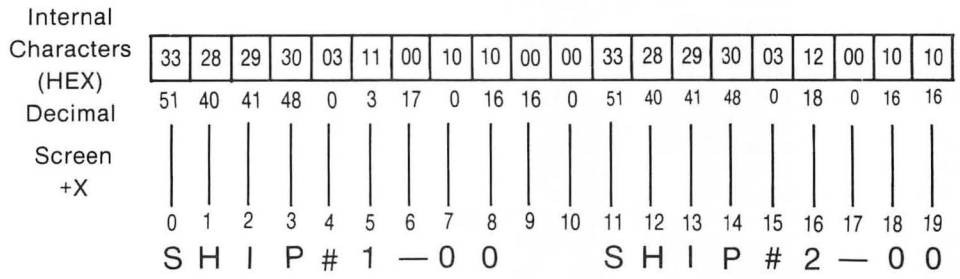


The explosion sound gives the effect of a slow rumble that slowly decreases in volume. We are working with a fairly low frequency and a distortion of zero in channel one and a distortion of 8 in channel 2. The use of two channels with these distortion and frequency values was determined experimentally to produce the best effect. ORing SBANG/4 with $E0 gives us a range of frequencies $E0-$EF. These are stored in AUDF1 and AUDF2. Since both AUDC1 and AUDC2 use the lower nibble for volume control, the value for AUDC1 can be obtained by ANDing the frequency with #$0F to mask out $E0. Since the upper nibble is the distortion, then a distortion of 8 can be obtained in AUDC2 by ORing AUDC1 with #$80. The equivalent sound routine in BASIC is as follows:

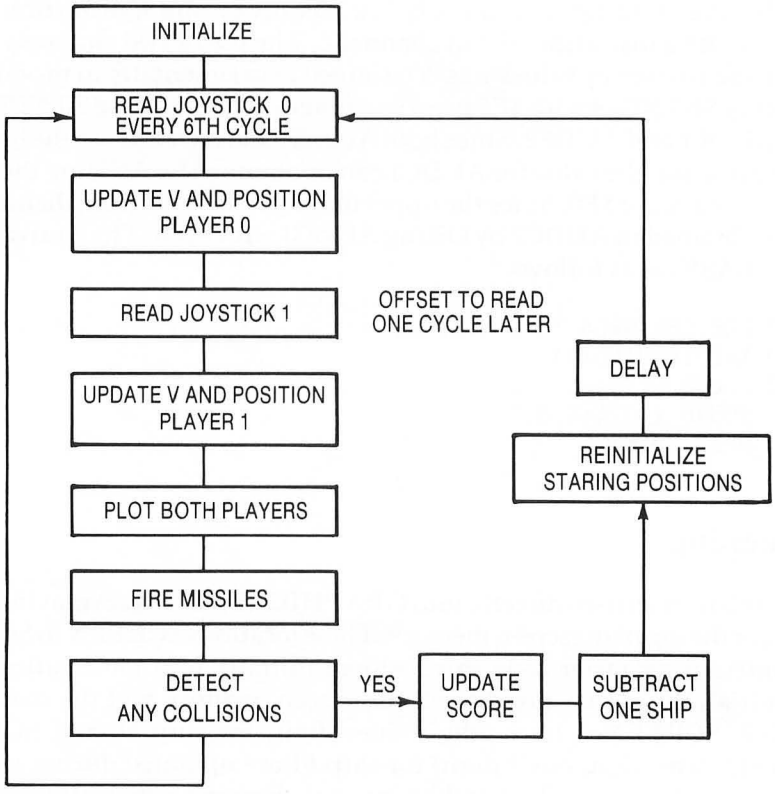
```
10 FOR SBANG=64 TO 0 STEP -1
20 X=INT(SBANG/4)
30 SOUND 0,224+X,0,X
40 SOUND 1,224+X,8,X
50 NEXT SBANG
```

Scorekeeping

The score line is written directly into GRAPHICS 1 (ANTIC 2) playfield memory beginning at the top of the screen memory. These locations SCREEN to SCREEN+19 contain internal character code valves stored initially in a table called SCLINE. During initialization they are moved into screen memory and the scoring digits, SCREEN+7, SCREEN+8 (tens digit, ones digit) for ship #0 and SCREEN+18, SCREEN+19 (tens digit, one's digit) for ship #1 are updtated during the game. A diagram showing the score line and its internal character data is illustrated below.



Flow Chart & Game Code




```
                00010 *SPACE WAR GAME - BY JEFFREY STANTON
                00020          .OR $4000
                00030 *ZERO PAGE EQUATES
00F0:           00040 SHPL     .EQ $F0
00F1:           00050 SHPH     .EQ $F1
00F2:           00060 SHPML    .EQ $F2
00F3:           00070 SHPMH    .EQ $F3
00F4:           00080 SHPMOL   .EQ $F4
00F5:           00090 SHPMOH   .EQ $F5
00F6:           00100 SL       .EQ $F6
00F7:           00110 SH       .EQ $F7
                00120 *PLAYER MISSILE EQUATES
D407:           00130 PMBASE   .EQ $D407
8800:           00140 PDATA    .EQ $8800
D01D:           00150 GRACTL   .EQ $D01D
022F:           00160 DMACTL   .EQ $22F
D008:           00170 SIZEP0   .EQ $D008
D009:           00180 SIZEP1   .EQ $D009
02C0:           00190 COLPM0   .EQ $2C0
02C1:           00200 COLPM1   .EQ $2C1
D000:           00210 HPOSP0   .EQ $D000
D001:           00220 HPOSP1   .EQ $D001
D004:           00230 HPOSM0   .EQ $D004
D005:           00240 HPOSM1   .EQ $D005
00C0:           00250 PMADR    .EQ $C0
D01E:           00260 HITCLR   .EQ $D01E
```

```
                00270 *COLLISIONS
D008:           00280 MOPL      .EQ $D008
D009:           00290 M1PL      .EQ $D009
D00C:           00300 POPL      .EQ $D00C
                00310 *MISC EQUATES
9400:           00320 NDLIST    .EQ $9400   ;ADR OF NEW DISPLAY LIST
9000:           00330 SCREEN    .EQ $9000
02C4:           00340 COLOR0    .EQ $2C4
02C6:           00350 COLOR2    .EQ $2C6
02C8:           00360 COLOR4    .EQ $2C8
E45C:           00370 SETVBK    .EQ $E45C
E462:           00380 XITVBK    .EQ $E462
0284:           00390 STRIG0    .EQ $284
0278:           00400 STICK     .EQ $278
D20A:           00410 RANDOM    .EQ $D20A
D200:           00420 AUDF1     .EQ $D200
D201:           00430 AUDC1     .EQ $D201
D202:           00440 AUDF2     .EQ $D202
D203:           00450 AUDC2     .EQ $D203
                00460 *DATA
4000: 08 08 08
4003: 1C 1C 1C
4006: 3E 3E 22
4009: 22 00 00 00470 SHAPE0    .HS 0808081C1C1C3E3E22220000
400C: 00 00 01
400F: 0E 1E 2E
4012: 04 04 08
4015: 00 00 00 00480           .HS 0000010E1E2E040408000000
4018: 00 00 00
401B: 70 3C 3F
401E: 3C 70 00
4021: 00 00 00 00490           .HS 000000703C3F3C7000000000
4024: 00 00 08
4027: 04 04 2E
402A: 1E 0E 01
402D: 00 00 00 00500           .HS 00000804042E1E0E01000000
4030: 00 22 22
4033: 3E 3E 1C
4036: 1C 1C 08
4039: 08 08 00 00510           .HS 0022223E3E1C1C1C08080800
403C: 00 00 08
403F: 10 10 3A
4042: 3C 38 40
4045: 00 00 00 00520           .HS 00000810103A3C3840000000
4048: 00 00 00
404B: 07 1E 7E
404E: 1E 07 00
4051: 00 00 00 00530           .HS 000000071E7E1E0700000000
4054: 00 00 40
4057: 38 3C 3A
405A: 10 10 08
405D: 00 00 00 00540           .HS 000040383C3A101008000000
4060: 00 0C 18
4063: 24 30 3C
4066: 48 54     00550 SHPL0    .HS 000C1824303C4854
4068: 02 02 01
406B: 02 03 00
406E: 02 01     00560 MSHAPE   .HS 0202010203000201
4070: 02 02 01
4073: 02 03 00
4076: 02 01     00570          .HS 0202010203000201
4078: 08 08 04
```

```
407B: 08 0C 00
407E: 08 04     00580           .HS 080804080C000804
4080: 08 08 04
4083: 08 0C 00
4086: 08 04     00590           .HS 080804080C000804
4088: 68 6A 6C
408B: 6E 70 72
408E: 74 76     00600 MISL0     .HS 686A6C6E70727476
4090: 78 7A 7C
4093: 7E 80 82
4096: 84 86     00610 MISL01    .HS 787A7C7E80828486
4098: 01 02 03
409B: 04 05 05
409E: 05 04 03
40A1: 02 01     00620 SKFLAG    .HS 0102030405050504030201
40A3: 70 70 70
40A6: 47 00 90
40A9: 02 02     00630 DLIST     .HS 7070704700900202
40AB: 02 02 02
40AE: 02 02 02
40B1: 02 02     00640           .HS 0202020202020202
40B3: 02 02 02
40B6: 02 02 02
40B9: 02 02     00650           .HS 0202020202020202
40BB: 02 02 02
40BE: 02 02 41
40C1: 00 94     00660           .HS 0202020202410094
40C3: 00 01 01
40C6: 01 00 FF
40C9: FF FF     00670 VTX       .HS 0001010100FFFFFF ;THRUST VECTORS
40CB: FF FF 00
40CE: 01 01 01
40D1: 00 FF     00680 VTY       .HS FFFF0001010100FF
40D3: 33 28 29
40D6: 30 03 11
40D9: 00 10 10
40DC: 00        00690 SCLINE    .HS 33282930031100101000 ;SCORE LINE DATA
40DD: 00 33 28
40E0: 29 30 03
40E3: 12 00 10
40E6: 10        00700           .HS 00332829300312001010
40E7: DD DD 6B
40EA: 6B AA AA
40ED: E6 E6     00710 DERES     .HS DDDD6B6BAAAAE6E6
40EF: 55 55 AA
40F2: AA 4A 4A
40F5: 35 35     00720           .HS 5555AAAA4A4A3535
40F7: 4A 4A 64
40FA: 64 92 92
40FD: 25 25     00730           .HS 4A4A646492922525
40FF: 11 11 48
4102: 48        00740           .HS 11114848
                00750 *VARIABLES
4103: 00        00760 VX        .DA #0      ;SHIP VELOCITY-HORIZ.
4104: 00        00770 VX1       .DA #0
4105: 00        00780 VY        .DA #0      ;SHIP VELOCITY-VERT.
4106: 00        00790 VY1       .DA #0
4107: 00        00800 VXP       .DA #0      ;SHIP VELOCITY-ALTERNATIVE
4108: 00        00810 VXP1      .DA #0
4109: 00        00820 VYP       .DA #0
410A: 00        00830 VYP1      .DA #0
410B:           00840 XPM0      .BS 1       ;CURRENT X POS SHIP
```

```
410C:          00850 XPM1      .BS 1
410D: 0A       00860 YOLDPM0   .DA #10      ;OLD Y POS SHIP
410E: 0A       00870 YOLDPM1   .DA #10
410F:          00880 YMISOLD0  .BS 1        ;OLD MISSILE Y POS
4110:          00890 YMISOLD1  .BS 1
4111:          00900 YPM0      .BS 1        ;CURRENT Y POS SHIP
4112:          00910 YPM1      .BS 1
4113: 00       00920 DIR       .DA #0       ;SHIP DIRECTION
4114: 00       00930 DIR1      .DA #0
4115:          00940 DIRM0     .BS 1        ;MISSILE TRAVEL DIRECTION
4116:          00950 DIRM1     .BS 1
4117: 00       00960 COUNT     .DA #0       ;COUNTER FOR JOYSTICK
4118: 00       00970 SCRCNTX   .DA #0       ;SCREEN X COUNTERS
4119: 01       00980 SCRCNTX1  .DA #1
411A: 00       00990 SCRCNTY   .DA #0       ;SCREEN Y COUNTERS
411B: 01       01000 SCRCNTY1  .DA #1
411C:          01010 FLAG      .BS 1        ;STICK BACK FLAG
411D:          01020 SKIPX     .BS 1        ;CONTROLS RATE OF SHIP UPDATE - HORIZ.
411E:          01030 SKIPX1    .BS 1
411F:          01040 SKIPY     .BS 1        ;CONTROLS RATE OF SHIP UPDATE - VERT.
4120:          01050 SKIPY1    .BS 1
4121:          01060 MASK      .BS 1
4122: FC       01070 MASK0     .DA #$FC     ;MISSILE MASKS
4123: F3       01080 MASK1     .DA #$F3
4124:          01090 XMIS0     .BS 1        ;HORIZ MISSILE POSITIONS
4125:          01100 XMIS1     .BS 1
4126:          01110 YMIS0     .BS 1        ;VERTICAL MISSILE POSITIONS
4127:          01120 YMIS1     .BS 1
4128:          01130 TMIS0     .BS 1        ;MISSILE TIMERS
4129:          01140 TMIS1     .BS 1
412A:          01150 SCOREOD   .BS 1        ;DIGITS
412B:          01160 SCOREOT   .BS 1        ;TENS
412C:          01170 SCORE1D   .BS 1
412D:          01180 SCORE1T   .BS 1
412E:          01190 EXCOUNT   .BS 1
412F:          01200 KILL0     .BS 1        ;KILL FLAG
4130:          01210 KILL1     .BS 1
4131:          01220 MBANG     .BS 2        ;SOUND COUNTERS-MISSILES
4133:          01230 SBANG     .BS 1        ;SOUND COUNTER
4134:          01240 DEREZ     .BS 12       ;TEMP STORAGE DURING DEREZ ROUTINE
               01250 START
               01260 *SETUP DLIST
4140: A9 00    01270           LDA #$00     ;NORMAL WIDTH
4142: 8D 09 D0 01280           STA SIZEP1
4145: A2 00    01290           LDX #$00
4147: BD A3 40 01300 DLOOP     LDA DLIST,X
414A: 9D 00 94 01310           STA NDLIST,X
414D: E8       01320           INX
414E: E0 20    01330           CPX #$20     ;32 ELEMENTS
4150: D0 F5    01340           BNE DLOOP
4152: A9 00    01350           LDA #NDLIST
4154: 8D 30 02 01360           STA 560
4157: A9 94    01370           LDA /NDLIST
4159: 8D 31 02 01380           STA 561
               01390 *CLEAR SCREEN - 1K
415C: A9 00    01400 CLRSCR    LDA #SCREEN ;SETUP POINTERS TO CLEAR SCREEN
415E: 85 F6    01410           STA SL
4160: A9 90    01420           LDA /SCREEN
4162: 85 F7    01430           STA SH
4164: A0 00    01440           LDY #$00
4166: 98       01450           TYA
4167: A2 04    01460           LDX #$04
```

```
4169: 91 F6    01470 .2       STA (SL),Y
416B: C8       01480          INY
416C: D0 FB    01490          BNE .2      ;CONTINUE UNTIL DONE WITH 256 BYTES
416E: E6 F7    01500          INC SH      ;DO NEXT PAGE
4170: CA       01510          DEX
4171: D0 F6    01520          BNE .2
               01530 *INITILIZE SCORE LINE
4173: A2 00    01540          LDX #$00
4175: BD D3 40 01550 .1       LDA SCLINE,X
4178: 9D 00 90 01560          STA SCREEN,X
417B: E8       01570          INX
417C: E0 14    01580          CPX #$14
417E: D0 F5    01590          BNE .1
4180: A9 00    01600          LDA #$00    ;ZERO OUT SCORE
4182: 8D 2A 41 01610          STA SCOREOD
4185: 8D 2B 41 01620          STA SCOREOT
4188: 8D 2C 41 01630          STA SCORE1D
418B: 8D 2D 41 01640          STA SCORE1T
418E: A9 28    01650          LDA #$28    ;YELLOW LETTERS
4190: 8D C4 02 01660          STA COLOR0
4193: A9 00    01670          LDA #$00    ;BLACK BACKGROUND & BORDER
4195: 8D C6 02 01680          STA COLOR2
4198: 8D C8 02 01690          STA COLOR4
419B: A9 00    01700 RESTART  LDA #$00
419D: 8D 2E 41 01710          STA EXCOUNT
41A0: 8D 2F 41 01720          STA KILL0
41A3: 8D 30 41 01730          STA KILL1
               01740 *INITILIZE P/M GRAPHICS
41A6: A9 88    01750          LDA #$88
41A8: 8D 07 D4 01760          STA PMBASE
41AB: A9 03    01770          LDA #$03    ;SET P/M GRAPHICS
41AD: 8D 1D D0 01780          STA GRACTL
41B0: A9 3E    01790          LDA #$3E    ;ENABLE P/M DMA SINGLE LINE
41B2: 8D 2F 02 01800          STA DMACTL
               01810 *INITILIZE SHIP 0
41B5: A9 00    01820          LDA #$00    ;NORMAL WIDTH
41B7: 8D 08 D0 01830          STA SIZEP0
41BA: A9 7A    01840          LDA #$7A    ;PLAYER #0 122 BLUE-LUM 10
41BC: 8D C0 02 01850          STA COLPM0
41BF: A9 50    01860          LDA #$50    ;INITIAL POS SHIP X=72
41C1: 8D 0B 41 01870          STA XPM0
41C4: 8D 00 D0 01880          STA HPOSP0  ; TELL ANTIC
41C7: A9 40    01890          LDA #$40    ;INITIAL POS SHIP Y=64
41C9: 18       01900          CLC
41CA: 69 25    01910          ADC #$25    ;32+CENTER PM SHAPE 5 BELOW TOP
41CC: 8D 11 41 01920          STA YPM0
               01930 *INITILIZE SHIP 1
41CF: A9 38    01940          LDA #$38    ;PLAYER #1 58 RED-LUM 8
41D1: 8D C1 02 01950          STA COLPM1
41D4: A9 A0    01960          LDA #$A0    ;INITIAL POS SHIP X=144
41D6: 8D 0C 41 01970          STA XPM1
41D9: 8D 01 D0 01980          STA HPOSP1
41DC: A9 60    01990          LDA #$60    ;INITIAL POS SHIP Y=96
41DE: 18       02000          CLC
41DF: 69 25    02010          ADC #$25
41E1: 8D 12 41 02020          STA YPM1
               02030 *CLEAR P/M AREA
41E4: A9 00    02040          LDA #$00    ;PDATAL
41E6: 85 C0    02050          STA PMADR
41E8: A9 88    02060          LDA /PDATA
41EA: 85 C1    02070          STA PMADR+1
41EC: A0 00    02080          LDY #$00
```

```
41EE: 98       02090          TYA
41EF: A2 08    02100          LDX #$08
41F1: 91 C0    02110 .3       STA (PMADR),Y
41F3: C8       02120          INY
41F4: D0 FB    02130          BNE .3
41F6: E6 C1    02140          INC PMADR+1 ;NEXT 256 BYTES
41F8: CA       02150          DEX
41F9: D0 F6    02160          BNE .3
               02170 *SET VBLANK
41FB: A9 07    02180          LDA #07
41FD: A2 42    02190          LDX /FRAME  ;HI BYTE VBLANK ROUTINE
41FF: A0 09    02200          LDY #FRAME  ;LO BYTE
4201: 20 5C E4 02210          JSR SETVBK
4204: A9 00    02220          LDA #$00
               02230 *READ STICK
4206: 4C 06 42 02240 FOREVER  JMP FOREVER
4209: EA       02250 FRAME    NOP
               02260 *READ STICK EVERY 6 TIMES
420A: EE 17 41 02270 CHKSTK   INC COUNT
420D: AD 17 41 02280          LDA COUNT
4210: C9 05    02290          CMP #$05    ;READ STICK ONLY EVERY 6TH TIME
4212: F0 03    02300          BEQ .4
4214: 4C 28 42 02310          JMP P1
               02320 *SETUP CALL TO JOYSTICK-PLAYER 0
4217: A2 00    02330 .4       LDX #$00    ;PLAYER 0
4219: BD 07 41 02340          LDA VXP,X   ;SET PLAYER'S NEW VELOCITY TO OLD VELOCITY
421C: 8D 03 41 02350          STA VX
421F: BD 09 41 02360          LDA VYP,X
4222: 8D 05 41 02370          STA VY
4225: 20 2C 43 02380          JSR JOYSTK
               02390 *UPDATE X & Y POSITIONS PLAYER #0
4228: A2 00    02400 P1       LDX #$00
422A: 20 A6 43 02410          JSR UPDATE
422D: 20 42 44 02420 .5       JSR PLOTSET0 ;PLOT PLAYER 0
4230: 20 82 44 02430          JSR PLOT
4233: A5 F4    02440          LDA SHPMOL  ;STORE AS OLD Y POS FOR NEXT CYCLE
4235: 8D 0D 41 02450          STA YOLDPM0
4238: AD 17 41 02460 CHKSTK1  LDA COUNT
423B: C9 06    02470          CMP #$06  READ STICK ONLY EVERY 6TH TIME
423D: F0 03    02480          BEQ .4
423F: 4C 58 42 02490          JMP P2
               02500 *SETUP CALL TO JOYSTICK-PLAYER 1
4242: A2 01    02510 .4       LDX #$01    ;PLAYER 1
4244: BD 07 41 02520          LDA VXP,X
4247: 8D 03 41 02530          STA VX
424A: BD 09 41 02540          LDA VYP,X
424D: 8D 05 41 02550          STA VY
4250: 20 2C 43 02560          JSR JOYSTK
4253: A9 00    02570          LDA #$00    ;RESET 4 CYCLE COUNTER
4255: 8D 17 41 02580          STA COUNT
               02590 *UPDATE X & Y POSITIONS PLAYER #1
4258: A2 01    02600 P2       LDX #$01
425A: 20 A6 43 02610          JSR UPDATE
425D: 20 62 44 02620 .5       JSR PLOTSET1 ;PLOT PLAYER 1
4260: 20 82 44 02630          JSR PLOT
4263: A5 F4    02640          LDA SHPMOL  ;STORE AS OLD Y POS FOR NEXT CYCLE
4265: 8D 0E 41 02650          STA YOLDPM1
               02660 *FIRE MISSILE
4268: A2 00    02670          LDX #$00
426A: 20 0B 45 02680          JSR MISSILE
426D: A2 01    02690          LDX #$01
426F: 20 0B 45 02700          JSR MISSILE
```

```
                02710 *CHECK COLLISION
                02720 *CHECK FOR OLD COLLISION FIRST
4272: AD 2F 41  02730           LDA KILLO   ;FIRST TEST FOR 2 PLAYER COLLISION
4275: C9 00     02740           CMP #$00
4277: F0 07     02750           BEQ .10     ;CAN'T BE 2 PLAYER COLLISION
4279: AD 30 41  02760           LDA KILL1
427C: C9 00     02770           CMP #$00
427E: D0 24     02780           BNE REMOVE
4280: AD 30 41  02790 .10       LDA KILL1   ;TEST FOR DERES PLAYER#1
4283: C9 00     02800           CMP #$00
4285: D0 31     02810           BNE REMOVE1
4287: AD 2F 41  02820           LDA KILLO   ;TEST FOR DERES PLAYER#0
428A: C9 00     02830           CMP #$00
428C: D0 3D     02840           BNE REMOVE0
                02850 *CHECK FOR NEW COLLISION
428E: AD 0C D0  02860           LDA P0PL    ;TEST P#0 AGAINST P#1
4291: D0 11     02870           BNE REMOVE
4293: AD 08 D0  02880           LDA M0PL    ;TEST M#0 AGAINST P#1
4296: C9 02     02890           CMP #$02
4298: F0 1E     02900           BEQ REMOVE1
429A: AD 09 D0  02910           LDA M1PL    ;TEST M#1 AGAINST P#0
429D: C9 01     02920           CMP #$01
429F: F0 2A     02930           BEQ REMOVE0
42A1: 4C DB 42  02940           JMP TESTE
42A4: A9 01     02950 REMOVE    LDA #01
42A6: 8D 2F 41  02960           STA KILL0
42A9: 8D 30 41  02970           STA KILL1
42AC: 20 DB 45  02980           JSR EXPLODE0
42AF: 20 E7 45  02990           JSR EXPLODE1
42B2: EE 2E 41  03000           INC EXCOUNT ;NEXT EXPLOSION FRAME
42B5: 4C DB 42  03010           JMP TESTE   ;TEST IF EXPLOSION FINISHED
42B8: A9 01     03020 REMOVE1   LDA #$01
42BA: 8D 30 41  03030           STA KILL1
42BD: A9 F0     03040           LDA #$F0    ;PUT MISSILE OFF SCREEN
42BF: 8D 05 D0  03050           STA HPOSM1
42C2: 20 E7 45  03060           JSR EXPLODE1
42C5: EE 2E 41  03070           INC EXCOUNT
42C8: 4C DB 42  03080           JMP TESTE
42CB: A9 01     03090 REMOVE0   LDA #$01
42CD: 8D 2F 41  03100           STA KILL0
42D0: A9 F0     03110           LDA #$F0    ;PUT MISSILE OFF SCREEN
42D2: 8D 04 D0  03120           STA HPOSM0
42D5: 20 DB 45  03130           JSR EXPLODE0
42D8: EE 2E 41  03140           INC EXCOUNT
42DB: AD 2E 41  03150 TESTE     LDA EXCOUNT ;TEST IF DONE WITH DERES CYCLE
42DE: C9 30     03160           CMP #$30
42E0: D0 41     03170           BNE ENDCYCLE
42E2: A9 00     03180           LDA #$00    ; SHUT OFF EXPLOSION SOUND
42E4: 8D 01 D2  03190           STA AUDC1
42E7: 8D 03 D2  03200           STA AUDC2
42EA: 8D 33 41  03210           STA SBANG
42ED: 20 39 46  03220           JSR SCORE   ;UPDATE SCORE
42F0: A9 00     03230           LDA #$00    ;RESET
42F2: 8D 2E 41  03240           STA EXCOUNT
42F5: 8D 2F 41  03250           STA KILL0
42F8: 8D 30 41  03260           STA KILL1
                03270 *REPOSITION SHIP AFTER KILL
42FB: A9 50     03280           LDA #$50    ;SHIP 0
42FD: 8D 0B 41  03290           STA XPM0
4300: 8D 00 D0  03300           STA HPOSP0
4303: A9 65     03310           LDA #$65
4305: 8D 11 41  03320           STA YPM0
```

```
4308: A9 A0    03330          LDA #$A0    ;SHIP 1
430A: 8D 0C 41 03340          STA XPM1
430D: 8D 01 D0 03350          STA HPOSP1
4310: A9 85    03360          LDA #$85
4312: 8D 12 41 03370          STA YPM1
4315: A9 00    03380          LDA #$00
4317: 8D 07 41 03390          STA VXP
431A: 8D 08 41 03400          STA VXP1
431D: 8D 09 41 03410          STA VYP
4320: 8D 0A 41 03420          STA VYP1
4323: 8D 1E D0 03430 ENDCYCLE STA HITCLR  ;WRITING ANYTHING CLEARS COLLISION REGISTER
4326: 20 8C 46 03440          JSR DOSOUND
4329: 4C 62 E4 03450          JMP XITVBK
               03460 *
               03470 *SUBROUTINE READ JOYSTICK
               03480 *INPUT X REG- # OF PLAYER
               03490 *VX,VY CURRENT PLAYER VELOCITY
               03500 *OUTPUT VXP,VYP PLAYER VELOCITY
432C: BD 78 02 03510 JOYSTK   LDA STICK,X
432F: 29 02    03520          AND #$02    ;DOWN BIT?
4331: D0 05    03530          BNE CHKLF
4333: A9 01    03540          LDA #01
4335: 8D 1C 41 03550          STA FLAG    ;YES STICK BACK
4338: BD 78 02 03560 CHKLF    LDA STICK,X
433B: 29 04    03570          AND #$04    ;LEFT BIT?
433D: D0 12    03580          BNE CHKRT
433F: DE 13 41 03590          DEC DIR,X
4342: BD 13 41 03600          LDA DIR,X
4345: C9 FF    03610          CMP #$FF
4347: D0 05    03620          BNE .1
4349: A9 07    03630          LDA #$07    ;SET DIR TO 7
434B: 9D 13 41 03640          STA DIR,X
434E: 4C 68 43 03650 .1       JMP CHKFD
4351: BD 78 02 03660 CHKRT    LDA STICK,X
4354: 29 08    03670          AND #$08    ;RIGHT BIT?
4356: D0 10    03680          BNE CHKFD
4358: FE 13 41 03690          INC DIR,X
435B: BD 13 41 03700          LDA DIR,X
435E: C9 08    03710          CMP #$08
4360: D0 05    03720          BNE .2
4362: A9 00    03730          LDA #$00    ;SET DIR TO 0
4364: 9D 13 41 03740          STA DIR,X
4367: EA       03750 .2       NOP
4368: BD 78 02 03760 CHKFD    LDA STICK,X
436B: 29 01    03770          AND #$01    ;UP BIT?
436D: D0 36    03780          BNE .9
               03790 *SET TO FREE FLOAT
436F: BD 13 41 03800          LDA DIR,X
4372: A8       03810          TAY
4373: 18       03820          CLC
4374: B9 C3 40 03830          LDA VTX,Y   ;GET X(DIR)THRUST VECTOR
4377: 6D 03 41 03840          ADC VX
437A: C9 FA    03850          CMP #$FA
437C: D0 02    03860          BNE .5
437E: A9 FB    03870          LDA #$FB    ;CLIP TO -5
4380: C9 06    03880 .5       CMP #$06
4382: D0 02    03890          BNE .6
4384: A9 05    03900          LDA #$05    ;CLIP TO 5
4386: 8D 03 41 03910 .6       STA VX      ;STORE OLD OR CLIPPED VALUE
4389: 9D 07 41 03920          STA VXP,X
438C: 18       03930          CLC
438D: B9 CB 40 03940          LDA VTY,Y   ;GET Y(DIR) THRUST VECTOR
```

```
4390: 6D 05 41 03950          ADC VY
4393: C9 FA    03960          CMP #$FA
4395: D0 02    03970          BNE .7
4397: A9 FB    03980          LDA #$FB    ;CLIP TO -5
4399: C9 06    03990 .7       CMP #$06
439B: D0 02    04000          BNE .8
439D: A9 05    04010          LDA #$05    ;CLIP TO 5
439F: 8D 05 41 04020 .8       STA VY      ;STORE OLD OR CLIPPED VALUE
43A2: 9D 09 41 04030          STA VYP,X
43A5: 60       04040 .9       RTS
               04050 *SUBROUTINE TO UPDATE X & Y POSITIONS
43A6: EA       04060 UPDATE   NOP
               04070 *UPDATE Y POS, VARIABLE # TIMES
               04080 *TEST WHEN TO UPDATE
43A7: 18       04090 UPDATEY  CLC
43A8: BD 09 41 04100          LDA VYP,X
43AB: 69 05    04110          ADC #$05    ;SO NEGATIVE #'S IN TABLE TOO
43AD: A8       04120          TAY
43AE: B9 98 40 04130          LDA SKFLAG,Y ;#TIMES TO SKIP IS RECIPRICAL OF SPEED
43B1: 9D 1F 41 04140          STA SKIPY,X
43B4: FE 1A 41 04150          INC SCRCNTY,X ;INCREMENT Y COUNTER
43B7: BD 1A 41 04160          LDA SCRCNTY,X
43BA: DD 1F 41 04170          CMP SKIPY,X ;IF MATCH UPDATE Y POS
43BD: B0 03    04180          BGE .1
43BF: 4C F1 43 04190          JMP UPDATEX
               04200 *UPDATE Y PLAYER POSITION
43C2: A9 00    04210 .1       LDA #$00    ;PERFORM Y UPDATE
43C4: 9D 1A 41 04220          STA SCRCNTY,X
43C7: BD 09 41 04230          LDA VYP,X
43CA: C9 00    04240          CMP #$00    ;IF HASN'T MOVED DON'T UPDATE
43CC: F0 23    04250          BEQ UPDATEX
43CE: 30 12    04260          BMI .2
43D0: FE 11 41 04270          INC YPMO,X
43D3: BD 11 41 04280          LDA YPMO,X
43D6: C9 E0    04290          CMP #$E0    ;HIT BOTTOM?
43D8: 90 05    04300          BLT .3
43DA: A9 18    04310          LDA #$18    ;WRAP TO TOP
43DC: 9D 11 41 04320          STA YPMO,X
43DF: 4C F1 43 04330 .3       JMP UPDATEX
43E2: DE 11 41 04340 .2       DEC YPMO,X
43E5: BD 11 41 04350          LDA YPMO,X
43E8: C9 18    04360          CMP #$18    ;HIT TOP?
43EA: B0 05    04370          BGE UPDATEX
43EC: A9 E0    04380          LDA #$E0    ;WRAP TO BOTTOM
43EE: 9D 11 41 04390          STA YPMO,X
               04400 *UPDATE X POS VARIABLE # TIMES
43F1: 18       04410 UPDATEX  CLC
43F2: BD 07 41 04420          LDA VXP,X
43F5: 69 05    04430          ADC #$05    ;SO NEG #'S IN TABLE TOO
43F7: A8       04440          TAY
43F8: B9 98 40 04450          LDA SKFLAG,Y
43FB: 9D 1D 41 04460          STA SKIPX,X
43FE: FE 18 41 04470          INC SCRCNTX,X
4401: BD 18 41 04480          LDA SCRCNTX,X
4404: DD 1D 41 04490          CMP SKIPX,X ;UPDATESCREEN POSITION EVERY SKIP TIME
4407: B0 03    04500          BGE .1
4409: 4C 41 44 04510          JMP EE
               04520 *UPDATE X PLAYER POSITION
440C: A9 00    04530 .1       LDA #$00
440E: 9D 18 41 04540          STA SCRCNTX,X
4411: BD 07 41 04550          LDA VXP,X
4414: C9 00    04560          CMP #$00
```

```
4416: F0 23    04570          BEQ .3
4418: 30 12    04580          BMI .2
441A: FE 0B 41 04590          INC XPM0,X
441D: BD 0B 41 04600          LDA XPM0,X
4420: C9 D8    04610          CMP #$D8    ;HIT RT SIDE?
4422: 90 05    04620          BLT .4
4424: A9 28    04630          LDA #$28    ;WRAP TO LEFT
4426: 9D 0B 41 04640          STA XPM0,X
4429: 4C 3B 44 04650 .4       JMP .3
442C: DE 0B 41 04660 .2       DEC XPM0,X
442F: BD 0B 41 04670          LDA XPM0,X
4432: C9 28    04680          CMP #$28    ;HIT LEFT?
4434: B0 05    04690          BGE .3
4436: A9 D8    04700          LDA #$D8    ;WRAP TO RIGHT
4438: 9D 0B 41 04710          STA XPM0,X
443B: BD 0B 41 04720 .3       LDA XPM0,X  ; NEW VALUE
443E: 9D 00 D0 04730          STA HPOSP0,X ; UPDATE ANTIC PLAYER HORIZ
4441: 60       04740 EE       RTS
               04750 *PLOT PLAYER0 SETUP
4442: AD 11 41 04760 PLOTSET0 LDA YPM0    ;CORRECTED YPOS
4445: 85 F2    04770          STA SHPML
4447: A9 88    04780          LDA /PDATA
4449: 18       04790          CLC
444A: 69 04    04800          ADC #$04    ;PLAYER0 IS 1K BEYOND START
444C: 85 F3    04810          STA SHPMH
444E: 85 F5    04820          STA SHPMOH
4450: AC 13 41 04830          LDY DIR
4453: B9 60 40 04840          LDA SHPL0,Y ;POINTER TO CORRECT SHAPE
4456: 85 F0    04850          STA SHPL
4458: A9 40    04860          LDA /SHAPE0 ;HI BYTE
445A: 85 F1    04870          STA SHPH
445C: AD 0D 41 04880          LDA YOLDPM0
445F: 85 F4    04890          STA SHPMOL
4461: 60       04900          RTS
               04910 *PLOT PLAYER1 SETUP
4462: AD 12 41 04920 PLOTSET1 LDA YPM1    ;CORRECTED YPOS
4465: 85 F2    04930          STA SHPML
4467: A9 88    04940          LDA /PDATA
4469: 18       04950          CLC
446A: 69 05    04960          ADC #$05    ;PLAYER1 IS 1.25K BEYOND START
446C: 85 F3    04970          STA SHPMH
446E: 85 F5    04980          STA SHPMOH
4470: AC 14 41 04990          LDY DIR1
4473: B9 60 40 05000          LDA SHPL0,Y ;POINTER TO CORRECT SHAPE
4476: 85 F0    05010          STA SHPL
4478: A9 40    05020          LDA /SHAPE0 ;HI BYTE ;BOTH PLAYER SHAPES SAME
447A: 85 F1    05030          STA SHPH
447C: AD 0E 41 05040          LDA YOLDPM1
447F: 85 F4    05050          STA SHPMOL
4481: 60       05060          RTS
               05070 *PUT SHAPE IN P/M AREA
4482: A0 00    05080 PLOT     LDY #$00    ;COUNTER
4484: A9 00    05090          LDA #$00    ;NEED 0 TO ERASE EACH TIME
4486: 91 F4    05100 .1       STA (SHPMOL),Y ;ERASE OLD SHAPE FIRST
4488: C8       05110          INY
4489: C0 0C    05120          CPY #$0C
448B: 90 F9    05130          BLT .1
448D: A0 00    05140          LDY #$00
448F: B1 F0    05150 .2       LDA (SHPL),Y  ;GET BYTE FROM PROPER SHAPE TABLE
4491: 91 F2    05160          STA (SHPML),Y ;PUT IN P/M AREA
4493: A5 F2    05170          LDA SHPML   ;TRANSFER NEW P/M POS TO OLD POS
```

```
4495: C8       05180          INY
4496: C0 0C    05190          CPY #$0C
4498: 90 F5    05200          BLT .2
449A: A5 F2    05210          LDA SHPML   ;TRANSFER NEW P/M POS TO OLD POS
449C: 85 F4    05220          STA SHPMOL
449E: A5 F3    05230          LDA SHPMH
44A0: 85 F5    05240          STA SHPMOH
44A2: 60       05250          RTS
               05260 *SETUP TO PLOT MISSILE 0
44A3: AD 26 41 05270 MISSET0  LDA YMIS0   ;MISSILE POSITION CORRECTED
44A6: 85 F2    05280          STA SHPML
44A8: A9 88    05290          LDA /PDATA
44AA: 18       05300          CLC
44AB: 69 03    05310          ADC #$03    ;MISSILES .75K BEYOND START
44AD: 85 F3    05320          STA SHPMH
44AF: 85 F5    05330          STA SHPMOH
44B1: AC 15 41 05340          LDY DIRM0
44B4: B9 88 40 05350          LDA MISL0,Y ;POINTER TO CORRECT MISSILE SHAPE
44B7: 85 F0    05360          STA SHPL
44B9: A9 40    05370          LDA /MSHAPE ;HI BYTE-BOTH P/M SHAPES SAME
44BB: 85 F1    05380          STA SHPH
44BD: AD 0F 41 05390          LDA YMISOLD0
44C0: 85 F4    05400          STA SHPMOL
44C2: AD 22 41 05410          LDA MASK0
44C5: 8D 21 41 05420          STA MASK
44C8: 20 B7 45 05430          JSR MPLOT
44CB: A5 F4    05440          LDA SHPMOL
44CD: 8D 0F 41 05450          STA YMISOLD0
44D0: AD 24 41 05460          LDA XMIS0
44D3: 8D 04 D0 05470          STA HPOSM0  ;MISSILE 0 HORIZ POS
44D6: 60       05480          RTS
               05490 *SETUP TO PLOT MISSILE 1
44D7: AD 27 41 05500 MISSET1  LDA YMIS1   ;MISSILE POSITION CORRECTED
44DA: 85 F2    05510          STA SHPML
44DC: A9 88    05520          LDA /PDATA
44DE: 18       05530          CLC
44DF: 69 03    05540          ADC #$03    ;MISSILES .75K BEYOND START
44E1: 85 F3    05550          STA SHPMH
44E3: 85 F5    05560          STA SHPMOH
44E5: AC 16 41 05570          LDY DIRM1
44E8: B9 90 40 05580          LDA MISL01,Y ;POINTER TO CORRECT MISSILE SHAPE
44EB: 85 F0    05590          STA SHPL
44ED: A9 40    05600          LDA /MSHAPE ;HI BYTE-BOTH P/M SHAPES SAME
44EF: 85 F1    05610          STA SHPH
44F1: AD 10 41 05620          LDA YMISOLD1
44F4: 85 F4    05630          STA SHPMOL
44F6: AD 23 41 05640          LDA MASK1
44F9: 8D 21 41 05650          STA MASK
44FC: 20 B7 45 05660          JSR MPLOT
44FF: A5 F4    05670          LDA SHPMOL
4501: 8D 10 41 05680          STA YMISOLD1
4504: AD 25 41 05690          LDA XMIS1
4507: 8D 05 D0 05700          STA HPOSM1  ;MISSILE 1 HORIZ POS
450A: 60       05710          RTS
               05720 *MISSILE SUBROUTINE -MISSILE TRACK ENDS AT COLLSION OR SCREEN ED(
450B: BD 84 02 05730 MISSILE  LDA STRIG0,X
450E: C9 00    05740          CMP #$00
4510: D0 37    05750          BNE .1
4512: BD 28 41 05760          LDA TMIS0,X
4515: C9 00    05770          CMP #$00
4517: D0 3A    05780          BNE .2
```

```
4519: A9 10    05790          LDA #$10    ; FLAG & INITIAL VOLUME FOR SHOT SOUND
451B: 9D 31 41 05800          STA MBANG,X
451E: FE 28 41 05810          INC TMISO,X ;INCREMENT TIMER
               05820 *ERASE OLD MISSILE & PUT AT SHIP
4521: BD 13 41 05830          LDA DIR,X
4524: 9D 15 41 05840          STA DIRMO,X ;DIR MISSILE TO MOVE THROUGHOUT TRAVEL
4527: BD 11 41 05850          LDA YPMO,X
452A: 18       05860          CLC
452B: 69 06    05870          ADC #$06    ;CORRECT TO SHIP CENTER
452D: 9D 26 41 05880          STA YMISO,X
4530: BD 0B 41 05890          LDA XPMO,X
4533: 18       05900          CLC
4534: 69 04    05910          ADC #$04    ;CORRECT TO SHIP CENTER
4536: 9D 24 41 05920          STA XMISO,X
4539: E0 00    05930          CPX #$00
453B: D0 06    05940          BNE .10
453D: 20 A3 44 05950          JSR MISSET0 ;TO MISSILE PLOT SETUP & PLOT
4540: 4C B6 45 05960          JMP EXIT
4543: 20 D7 44 05970 .10      JSR MISSET1
4546: 4C B6 45 05980          JMP EXIT
4549: BD 28 41 05990 .1       LDA TMISO,X
454C: C9 00    06000          CMP #$00
454E: D0 03    06010          BNE .2
4550: 4C B6 45 06020          JMP EXIT
               06030 *MOVE MISSILE IN PROPER DIRECTION
4553: BC 15 41 06040 .2       LDY DIRMO,X
4556: B9 C3 40 06050          LDA VTX,Y
4559: 0A       06060          ASL         ;DOUBLE VELOCITY VECTOR
455A: 18       06070          CLC
455B: 7D 24 41 06080          ADC XMISO,X
455E: 9D 24 41 06090          STA XMISO,X
4561: B9 CB 40 06100          LDA VTY,Y
4564: 0A       06110          ASL         ;DOUBLE VELOCITY VECTOR
4565: 18       06120          CLC
4566: 7D 26 41 06130          ADC YMISO,X
4569: 9D 26 41 06140          STA YMISO,X
               06150 *HAS MISSILE HIT SCREEN EDGE
456C: C9 20    06160          CMP #$20
456E: B0 03    06170          BGE .3
4570: 4C 9F 45 06180          JMP ERASEM
4573: C9 D8    06190 .3       CMP #$D8
4575: 90 03    06200          BLT .4
4577: 4C 9F 45 06210          JMP ERASEM
457A: BD 24 41 06220 .4       LDA XMISO,X
457D: C9 30    06230          CMP #$30
457F: B0 03    06240          BGE .5
4581: 4C 9F 45 06250          JMP ERASEM
4584: C9 D0    06260 .5       CMP #$D0
4586: 90 03    06270          BLT .6
4588: 4C 9F 45 06280          JMP ERASEM
458B: EA       06290 .6       NOP
458C: FE 28 41 06300          INC TMISO,X
               06310 *ERASE & REPLOT MISSILE
458F: E0 00    06320 ERASE    CPX #$00
4591: D0 06    06330          BNE .8
4593: 20 A3 44 06340          JSR MISSET0
4596: 4C B6 45 06350          JMP EXIT
4599: 20 D7 44 06360 .8       JSR MISSET1
459C: 4C B6 45 06370          JMP EXIT
               06380 *ERASE MISSILE OFF SCREEN
459F: A9 E0    06390 ERASEM   LDA #$E0
```

```
45A1: 9D 24 41 06400          STA XMISO,X ;PLOT OFF SCREEN
45A4: E0 00    06410          CPX #$00
45A6: D0 06    06420          BNE .71
45A8: 20 A3 44 06430          JSR MISSET0
45AB: 4C B1 45 06440          JMP .7
45AE: 20 D7 44 06450 .71      JSR MISSET1
45B1: A9 00    06460 .7       LDA #$00
45B3: 9D 28 41 06470          STA TMISO,X
45B6: 60       06480 EXIT     RTS
               06490 *PUT MISSILE SHAPE IN P/M AREA SUBROUTINE
45B7: A0 00    06500 MPLOT    LDY #$00
45B9: B1 F4    06510 .1       LDA (SHPMOL),Y ;LOAD OLD SHAPE
45BB: 2D 21 41 06520          AND MASK    ;MASK WITH PROPER MISSILE BEING MOVED
45BE: 91 F4    06530          STA (SHPMOL),Y ;STOREING IT DOESN'T ERASE OTHER MISSILES
45C0: C8       06540          INY
45C1: C0 02    06550          CPY #$02
45C3: 90 F4    06560          BLT .1
45C5: A0 00    06570          LDY #$00
45C7: B1 F0    06580 .2       LDA (SHPL),Y  ;GET BYTE FROM CORRECT MISSILE SHAPE TABLE
45C9: 11 F2    06590          ORA (SHPML),Y ;OR AGAINST CURRENT OTHER MISSILES
45CB: 91 F2    06600          STA (SHPML),Y ;PUT IN MISSILE AREA
45CD: C8       06610          INY
45CE: C0 02    06620          CPY #$02
45D0: 90 F5    06630          BLT .2
45D2: A5 F2    06640          LDA SHPML    ;TRANSFER TO OLD POINTERS
45D4: 85 F4    06650          STA SHPMOL
45D6: A5 F3    06660          LDA SHPMH
45D8: 85 F5    06670          STA SHPMOH
45DA: 60       06680          RTS
               06690 *SUBROUTINES TO EXPLODE SHIPS
45DB: 20 42 44 06700 EXPLODE0 JSR PLOTSET0
45DE: 20 F3 45 06710          JSR EXPLODE
45E1: A5 F4    06720          LDA SHPMOL
45E3: 8D 0D 41 06730          STA YOLDPM0
45E6: 60       06740          RTS
45E7: 20 62 44 06750 EXPLODE1 JSR PLOTSET1
45EA: 20 F3 45 06760          JSR EXPLODE
45ED: A5 F4    06770          LDA SHPMOL
45EF: 8D 0E 41 06780          STA YOLDPM1
45F2: 60       06790          RTS
               06800 *DE-RES SHAPE
45F3: A0 00    06810 EXPLODE  LDY #$00    ;COUNTER
45F5: A9 00    06820          LDA #$00    ;NEED 0 TO ERASE EACH TIME
45F7: 91 F4    06830 .1       STA (SHPMOL),Y ;ERASE OLD SHAPE FIRST
45F9: C8       06840          INY
45FA: C0 0C    06850          CPY #$0C
45FC: 90 F9    06860          BLT .1
45FE: AE 2E 41 06870          LDX EXCOUNT ;COUNTER
4601: A0 00    06880          LDY #$00    ;START WITH 0TH BYTE IN SHAPE
4603: E0 00    06890 .2       CPX #$00    ;FIRST TIME?
4605: D0 05    06900          BNE .22
4607: B1 F0    06910          LDA (SHPL),Y;GET BYTE FROM PROPER SHAPE TABLE
4609: 99 34 41 06920          STA DEREZ,Y ;DO THIS ONLY FIRST TIME
460C: AD 0A D2 06930 .22      LDA RANDOM
460F: 0D 0A D2 06940          ORA RANDOM  ;DEGRADE IMAGE RANDONILY
4612: 39 34 41 06950          AND DEREZ,Y
4615: 99 34 41 06960          STA DEREZ,Y ;TEMP STORE DEGRADED IMAGE
4618: B1 F0    06970          LDA (SHPL),Y
461A: 2D 0A D2 06980          AND RANDOM  ;DEGRADE IMAGE RANDOMILY
461D: 2D 0A D2 06990          AND RANDOM
4620: 2D 0A D2 07000          AND RANDOM
```

```
4623: 2D 0A D2 07010          AND RANDOM
4626: 19 34 41 07020          ORA DEREZ,Y ;COMBINE 2 DEGRADED IMAGES SO LESS DEGRADED
4629: 91 F2    07030          STA (SHPML),Y ;PUT IN P/M AREA
462B: C8       07040          INY         ;NEXT BYTE IN SHAPE
462C: C0 0C    07050          CPY #$0C    ;DONE?
462E: 90 D3    07060          BLT .2
4630: A5 F2    07070          LDA SHPML   ;TRANSFER NEW P/M POS TO OLD POS
4632: 85 F4    07080          STA SHPMOL
4634: A5 F3    07090          LDA SHPMH
4636: 85 F5    07100          STA SHPMOH
4638: 60       07110          RTS
               07120 *SCORE SUBROUTINE
4639: AD 2F 41 07130 SCORE    LDA KILL0
463C: F0 12    07140          BEQ .1
463E: EE 2A 41 07150          INC SCOREOD ;INC DIGITS #0
4641: AD 2A 41 07160          LDA SCOREOD ;TEST IF >9
4644: C9 0A    07170          CMP #$0A
4646: D0 08    07180          BNE .1
4648: EE 2B 41 07190          INC SCOREOT ;INC TENS #0
464B: A9 00    07200          LDA #$00
464D: 8D 2A 41 07210          STA SCOREOD ;ZERO DIGITS #0
4650: AD 30 41 07220 .1       LDA KILL1
4653: F0 12    07230          BEQ .2
4655: EE 2C 41 07240          INC SCORE1D ;INC DIGITS #1
4658: AD 2C 41 07250          LDA SCORE1D
465B: C9 0A    07260          CMP #$0A
465D: D0 08    07270          BNE .2
465F: EE 2D 41 07280          INC SCORE1T
4662: A9 00    07290          LDA #$00
4664: 8D 2C 41 07300          STA SCORE1D
4667: AD 2B 41 07310 .2       LDA SCOREOT
466A: 18       07320          CLC
466B: 69 10    07330          ADC #$10
466D: 8D 07 90 07340          STA SCREEN+7 ;PLACE IN SCREEN MEMORY
4670: AD 2A 41 07350          LDA SCOREOD
4673: 18       07360          CLC
4674: 69 10    07370          ADC #$10
4676: 8D 08 90 07380          STA SCREEN+8
4679: AD 2D 41 07390          LDA SCORE1T
467C: 18       07400          CLC
467D: 69 10    07410          ADC #$10
467F: 8D 12 90 07420          STA SCREEN+18
4682: AD 2C 41 07430          LDA SCORE1D
4685: 18       07440          CLC
4686: 69 10    07450          ADC #$10
4688: 8D 13 90 07460          STA SCREEN+19
468B: 60       07470          RTS
               07480 DOSOUND
468C: AD 2E 41 07490          LDA EXCOUNT ; SHIP EXPLODING?
468F: F0 24    07500          BEQ MSOUND  ; NO? THEN GO CHECK FOR SHOT SOUNDS
4691: C9 01    07510          CMP #$01    ; IS IT 1ST TIME
4693: D0 05    07520          BNE .1
4695: A9 40    07530          LDA #$40    ;THIS WILL BE DIVIDED BY 4 SO
4697: 8D 33 41 07540          STA SBANG   ;VOLUME GOES FROM $10 TO 0 IN STEPS OF
                                            EVERY 4 VBLANKS
469A: CE 33 41 07550 .1       DEC SBANG
469D: AD 33 41 07560          LDA SBANG
46A0: 4A       07570          LSR
46A1: 4A       07580          LSR
46A2: 09 E0    07590          ORA #$E0    ; LOWER IT
46A4: 8D 00 D2 07600          STA AUDF1   ;AUDF1
46A7: 8D 02 D2 07610          STA AUDF2   ;AUDF2
```

```
46AA: 29 0F    07620          AND #$0F    ;KEEP VOLUME
46AC: 8D 01 D2 07630          STA AUDC1   ;DISTORTION 0
46AF: 09 80    07640          ORA #$80    ;DISTORTION 8
46B1: 8D 03 D2 07650          STA AUDC2
46B4: 60       07660          RTS
               07670 MSOUND
46B5: A2 01    07680          LDX #$01
46B7: A0 02    07690          LDY #$02    ;AUDC1 & AUDC2 ARE 2 BYTES APART
46B9: BD 31 41 07700 .1       LDA MBANG,X
46BC: F0 0E    07710          BEQ .2      ;IF 0 THEN NO SOUND
46BE: DE 31 41 07720          DEC MBANG,X ; LET IT COUNTDOWN TO 0 EVERY CYCLE
46C1: BD 31 41 07730          LDA MBANG,X
46C4: 99 01 D2 07740          STA AUDC1,Y ; AUDC
46C7: A9 00    07750          LDA #$00
46C9: 99 00 D2 07760          STA AUDF1,Y
46CC: 88       07770 .2       DEY
46CD: 88       07780          DEY
46CE: CA       07790          DEX
46CF: 10 E8    07800          BPL .1
46D1: 60       07810          RTS
```

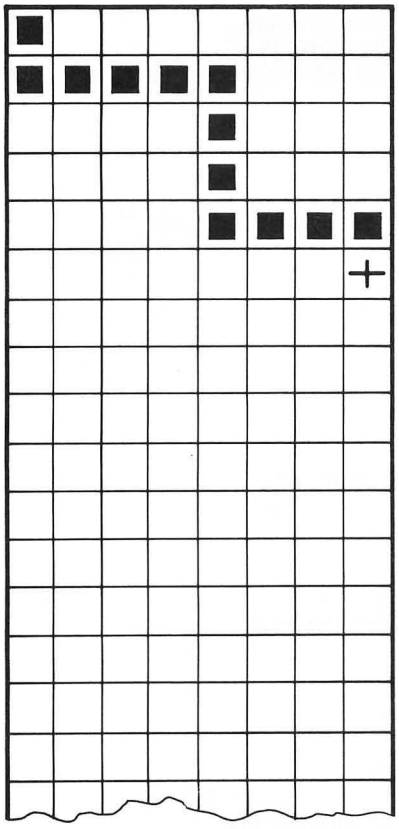
Player Missile Editor

We have included a player-missile editor that will make designing player shapes easy. The grid on the left hand side of the screen supports a shape twenty-two scan lines high in the single-resolution mode. Each of the enlarged blocks is the same shape as the individual pixels on the screen.

The cursor appears in the upper left hand corner. The message at the upper right initially is set in the NOPLOT mode. To put the cursor in the PLOT mode, press the letter P, for the ERASE mode, press the letter E, or return to the NOPLOT mode by pressing N. Pressing any of the arrow keys without the control key moves the cursor. Move the cursor to your desired starting position, and press P for plot. A block fills for each position of the shape, and the byte's decimal and hexadecimal value appears to the side.

As your shape forms, you will begin to see an actual sized player shape emerge in the lower right portion of the screen. You can toggle the player width by pressing the W key. It will progress from single width to double width to quadruple width before returning to single width. Likewise, toggle the player resolution from single-line resolution to double-line resolution by pressing the R key. If you wish to clear the entire screen of your shape, just press SHIFT CLEAR.

The editor, which is written in BASIC, is somewhat slow. Most of its slowness is due to our decision to write it in graphics mode 8, so that we could plot accurately while simutaneously incorporating text on the screen. Machine language programmers need the hexadecimal values of the player shapes for their shape tables. Owners of the ABC BASIC compiler by Monarch Data Systems will find that the program compiles with no modification,and runs significantly faster.



128	80	PLOT
248	F8	
008	08	
008	08	
015	0F	

```
5 REM PLAYER MISSILE SHAPE EDITOR Jeff Stanton
10 POKE 106,PEEK(106)-16:GRAPHICS 8+16
11 POKE 559,0
12 GOSUB 1000
13 GOSUB 2000
14 POKE 559,62
15 DIM MG(8,22),VALUE$(22),VALUE(22),V$(3),T$(3),W$(6),VH$(1),VL$(1),VT$(2)
16 VALUE$="":VALUE$(22)="":VALUE$(2)=VALUE$
17 POKE 710,0:GOSUB 815
77 REM READ KEYBOARD
78 OPEN #2,4,0,"K:"
79 ZZ=USR(ADR(PMPOKE$),ADR(VALUE$),PLAYER0,RES)
80 CH=1:GET #2,CHR:X0=X:Y0=Y
81 IF CHR=45 THEN 205:REM UP
85 IF CHR=61 THEN 225:REM DOWN
90 IF CHR=43 THEN 245:REM LEFT
95 IF CHR=42 THEN 265:REM RIGHT
100 IF CHR=80 THEN 125:REM P
105 IF CHR=69 THEN 145:REM E
110 IF CHR=78 THEN 135:REM N
111 IF CHR=82 THEN 1100:REM R
112 IF CHR=87 THEN 1200:REM W
113 IF CHR=125 THEN 3000:REM }
115 GOTO 79
```

```
120 REM SET FLAGS FOR P,E,N KEYS
125 PLT=1:ERASE=0:W$="PLOT  ":GOSUB 700:GOTO 330
135 PLT=0:ERASE=0:W$="NOPLOT":GOSUB 700:GOTO 79
145 PLT=0:ERASE=1:W$="ERASE ":GOSUB 700:GOTO 365
200 REM MOVE CURSOR UP
205 Y=Y-1:IF Y=0 THEN CH=0
210 IF Y=0 THEN Y=1
215 GOTO 300
220 REM MOVE CURSOR DOWN
225 Y=Y+1:IF Y=23 THEN CH=0
230 IF Y=23 THEN Y=22
235 GOTO 300
240 REM MOVE CURSOR LEFT
245 X=X-1:IF X=0 THEN CH=0
250 IF X=0 THEN X=1
255 GOTO 300
260 REM MOVE CURSOR RIGHT
265 X=X+1:IF X=9 THEN CH=0
270 IF X=9 THEN X=8
275 GOTO 300
300 REM CHANGE MATRIX AND PLOT OR ERASE AS NEEDED
305 IF CH=0 THEN 79
310 IF PLT=1 THEN 330
315 IF ERASE=1 THEN 365
320 GOSUB 400:GOTO 79
325 REM PLOT BLOCK AND INVERSE CURSOR
330 FOR I=1 TO 7:FOR J=1 TO 5:BX=18+X*11:BY=Y*8
345 PLOT BX+I,BY+J:NEXT J:NEXT I
355 MG(X,Y)=1:GOSUB 400:GOTO 500
360 REM ERASE BLOCK
365 COLOR 0:FOR I=1 TO 7:FOR J=1 TO 5
380 BX=18+X*11:BY=Y*8
385 PLOT BX+I,BY+J:NEXT J:NEXT I:COLOR 1
396 MG(X,Y)=0:GOSUB 400:GOTO 500
400 REM PLOT NEW CURSOR POSITION AFTER ERASING OLD
405 C=MG(XO,YO):IF C=1 THEN 415
410 COLOR 0
415 CX=19+XO*11:CY=3+YO*8
425 PLOT CX,CY:DRAWTO CX+4,CY
430 PLOT CX+2,CY-2:DRAWTO CX+2,CY+2
435 COLOR 1:C=MG(X,Y):IF C=0 THEN 445
440 COLOR 0
445 CX=19+X*11:CY=3+Y*8
455 PLOT CX,CY:DRAWTO CX+4,CY
460 PLOT CX+2,CY-2:DRAWTO CX+2,CY+2
465 COLOR 1:RETURN
500 REM CALCULATE NEW VALUE ROW
510 IF PLT=0 AND ERASE=0 THEN 79
515 VALUE(Y)=MG(1,Y)*128+MG(2,Y)*64+MG(3,Y)*32+MG(4,Y)*16+MG(5,Y)*8
516 VALUE(Y)=VALUE(Y)+MG(6,Y)*4+MG(7,Y)*2+MG(8,Y)
517 VALUE$(Y)=CHR$(VALUE(Y))
520 GOSUB 600:GOTO 79
600 REM POKES OR PRINTS VALUE ON SCREEN
605 YP=Y*8:XP=17:V$="000":T$=STR$(VALUE(Y))
619 REM GIVES A THREE CHARACTER NUMBER WITH LEADING 0'S
620 V$(4-LEN(T$))=T$
625 FOR U=1 TO 3
629 REM FIND POSITION IN CHARACTER SET
630 I2=57344+((ASC(V$(U,U))-32)*8)
635 I3=I1+YP*40+XP+U-1
639 REM POKE CHARACTER ON SCREEN (8 BYTES DEEP)
640 FOR Z=0 TO 7:POKE I3+Z*40,PEEK(I2+Z):NEXT Z
```

```
645 NEXT U
650 REM PLOTS TWO DIGIT HEX VALUE OF ROW
652 XP=22:V1=INT(VALUE(Y)/16)
654 V2=VALUE(Y)-(16*V1)
656 IF V1=10 THEN VH$="A"
658 IF V1=11 THEN VH$="B"
660 IF V1=12 THEN VH$="C"
662 IF V1=13 THEN VH$="D"
664 IF V1=14 THEN VH$="E"
666 IF V1=15 THEN VH$="F"
668 IF V1<10 THEN VH$=STR$(V1)
670 IF V2=10 THEN VL$="A"
672 IF V2=11 THEN VL$="B"
674 IF V2=12 THEN VL$="C"
676 IF V2=13 THEN VL$="D"
678 IF V2=14 THEN VL$="E"
680 IF V2=15 THEN VL$="F"
682 IF V2<10 THEN VL$=STR$(V2)
686 VT$=VH$:VT$(LEN(VT$)+1)=VL$
687 FOR U=1 TO 2
689 I2=57344+((ASC(VT$(U,U))-32)*8)
690 I3=I1+YP*40+XP+U-1
695 FOR Z=0 TO 7:POKE I3+Z*40,PEEK(I2+Z):NEXT Z
697 NEXT U
699 RETURN
700 REM PRINTS WORDS ERASE, NOPLOT, & PLOT AT BOTTOM
705 YP=16:XP=32:FOR U=1 TO 6
715 I2=57344+((ASC(W$(U,U))-32)*8)
720 I3=I1+YP*40+XP+U-1
725 FOR Z=0 TO 7:POKE I3+Z*40,PEEK(I2+Z):NEXT Z
730 NEXT U:RETURN
815 REM INITILIZE AND DRAW GRID
817 IO=PEEK(560)+PEEK(561)*256:I1=PEEK(IO+4)+PEEK(IO+5)*256
818 PLT=0:ERASE=0
819 W$="NOPLOT":GOSUB 700
820 REM DRAW GRID
825 COLOR 1
830 FOR J=1 TO 9
835 X=16+11*J:PLOT X,7:DRAWTO X,183
840 NEXT J
845 FOR I=1 TO 23
850 Y=-1+8*I:PLOT 26,Y:DRAWTO 114,Y
855 NEXT I
860 REM CLEAR MATRIX
865 FOR I=1 TO 8:FOR J=1 TO 22:MG(I,J)=0:NEXT J:NEXT I
867 FOR J=1 TO 22:VALUE(J)=0:NEXT J
869 REM PLOT INITIAL CURSOR POSITION
870 X=1:Y=1
872 PLOT 30,11:DRAWTO 34,11
873 PLOT 32,9:DRAWTO 32,13
875 RETURN
1000 REM INITIALIZE PMG
1010 PMBASE=PEEK(106):SDMCTL=559:SINGLE=62:DOUBLE=46
1020 POKE 53277,3:POKE 54279,PMBASE:POKE 704,200:POKE 53248,175
1030 REM CLEAR PLAYER 0 RAM
1040 DATA 104,104,133,207,104,133,206,169,0,170,168,145,206,200,192
1042 DATA 0,208,249,230,207,232,224,4,208,242,96
1045 DIM CLEAR$(26):FOR L1=1 TO 26:READ X:CLEAR$(L1,L1)=CHR$(X):NEXT L1
1050 X=USR(ADR(CLEAR$),PMBASE*256+1024)
1060 RETURN
1100 REM CHANGE PLAYER FROM SINGLE TO DOUBLE LINE RESOLUTION
1110 RES=ABS(RES-2)+1
```

```
1120 ZZ=USR(ADR(PMPOKE$),ADR(VALUE$),PLAYER0,RES)
1130 GOTO 79
1200 REM CHANGE PLAYER0 WIDTH
1210 WIDTH=WIDTH+1:IF WIDTH>3 THEN WIDTH=0
1220 IF WIDTH=2 THEN WIDTH=3
1230 POKE 53256,WIDTH:GOTO 79
2000 DATA 104,104,133,207,104,133,206,104,133,209,104,133,208,104
2005 DATA 104,141,0,6,169,0,168,145,208,200,192
2010 DATA 44,208,249,160,0,141,2,6,141,3,6,162,0,172,2,6,177
2015 DATA 206,172,3,6,145,208,232,238
2020 DATA 3,6,236,0,6,208,237,238,2,6,173,2,6,201,22,208,225,96
2030 DIM PMPOKE$(68):FOR L1=1 TO 68:READ X:PMPOKE$(L1,L1)=CHR$(X):NEXT L1
2040 RES=1:PLAYER0=PMBASE*256+1124
2050 RETURN
3000 REM RECLEAR SCREEN
3005 ? #6;"}";
3010 VALUE$="":VALUE$(22)="":VALUE$(2)=VALUE$
3020 GOSUB 815
3030 GOTO 79
```

Player-Missile Movement Using Strings

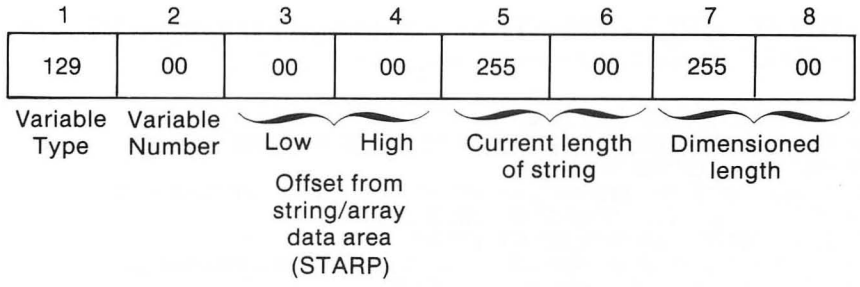
An alternate method of moving player shapes vertically at close to Machine language speed is to take advantage of Atari BASIC's string manipulation routines. One in particular in the form of P0$(N)=S0$ will take a smaller string such as S0$="ABCD" and place it starting at the Nth position of the larger string P0$. Thus, if N=6, P0$ would be as follows:

BYTE	VALUE
1	0
2	0
3	0
4	0
5	0
6	ASCII A
7	ASCII B
8	ASCII C
9	ASCII D
10	0
11	0
.	.
.	.

The shorter string could just as easily be our player shape. It would be very easy to move it just by changing the value of N. The only hitch is that we would have to erase its previous position before we moved it, or the player string would begin to repeat itself, or parts of itself, within the larger (256 byte) player area string. Since there is no need to move the player more than two bytes at a time, we can easily accomplish the erase by overwriting the trailing or leading edge of our last position with two trailing null or zero characters at each end.

How BASIC Stores Strings

The biggest problem in this method is to fool Atari BASIC into assigning the player-missile display area as a string variable. In order to understand how to do this, it is necessary to know how Atari BASIC stores variables in memory. Two areas are set aside in memory. The first, the Variable Value Table Pointer (VVTP), stores eight bytes of information for each variable declared in your BASIC program. The second, the String Array Table, reserves space in memory according to the size specified when you dimension an array. The VVTP table is arranged as in the following example. If it were for a string dimensioned as P0$(255) it would have these values:



Every time BASIC has to access information in a string it goes to the VVPT to find the current address of the string. Byte 1 contains the variable type which is 129 for a string. Byte 2 contains the variable number. BASIC tokenizes variables and refers to them by numbers, not names. Bytes 3 and 4 contain the value of the offset from the string/array area (STARP) where BASIC reserves memory for all dimensioned variables. The values are in low byte, high byte order. Thus the address of your string is STARP + Byte 3 + 256*Byte 4.

If we change the pointers to bytes 3 and 4 so that they now point to the player-missile area for player #0, we could use string move commands to move our player. To do this we will need to calculate the difference between the player-missile area and where BASIC was planning to store our string in STARP.

```
VVTP= PEEK(134) + 256*PEEK(135)
STARP = PEEK(140) + 256*PEEK(141)

PLAYER0 =PMBASE*256 + 1024
SO
OFFSET=PLAYER0-STARP
```

If we take this OFFSET value and divide it into its low and high byte values, then POKE them into bytes 3 and 4, BASIC would think that our strings were actually in the player-missile area. Since we have four players, we have to do this for each group of eight bytes in the VVTP table. Your dimensioned player strings should be the first variables in the program; otherwise, you will have to scan through the table looking for a match.

The following example uses this technique to move four players vertically. Horizontal movement is of course done by setting the player's horizontal position register. The example has four randomly moving, bouncing balls which reverse direction upon collision with another ball or the playfield boundary.

Just to give you a rough idea of the actual memory locations and the change we need to make in the VVTP table for player #0, the values for the example are listed below:

VVTP =7760
STARP =10932
PMBASE = 38912 (High byte = 152)
PLAYER0 = 39936

so that OFFSET = PLAYER0–STARP = 29004 then LO = 76 and HI = 113 and they are POKED into bytes 3 and 4 respectively.

```
10 REM DAN PINAL'S PM-STRING DEMO
20 REM FOR SANITY'S SAKE IF YOU INTEND TO OFFSET A STRING FROM ITS
25 REM ASSIGNED LOCATION REMEMBER:
30 REM 1: MAKE SURE THE STRING'S OFFSET WILL BE THE FIRST ONES FOUND
35 REM IN THE TABLE I.E. DIMENSION THEM FIRST
40 REM 2: OFFSET THE STRINGS BEFORE YOU PUT DATA IN THEM
50 REM 3: IF FOR SOME SOME REASEON YOU BLOW IT LIST THE PROGRAM TO
55 REM THE DISK,POWER DOWN AND ENTER IT BACK
60 REM START
70 CLR
80 DIM PO$(256),P1$(256),P2$(256),P3$(256)
90 DIM SO$(16),S1$(16),S2$(16),S3$(16),X(3),Y(3),DX(3),DY(3)
100 POKE 106,PEEK(106)-8:GRAPHICS 3+16:PMBASE=PEEK(106):PLAYER0=PMBASE*256+1024
110 DLIST=PEEK(560)+256*PEEK(561)
120 SCREEN=PEEK(DLIST+4)+256*PEEK(DLIST+5)
130 STARP=PEEK(140)+256*PEEK(141)
140 VVTP=PEEK(134)+256*PEEK(135)
150 OFFSET=PLAYER0-STARP
160 REM TRICKS IT INTO PM AREA
170 FOR L1=2 TO 26 STEP 8
180 HI=INT(OFFSET/256):LO=OFFSET-HI*256
190 POKE VVTP+L1,LO:POKE VVTP+L1+1,HI
200 OFFSET=OFFSET+256
210 NEXT L1
220 REM TURN ON PLAYERS BUT TURN OFF PLAYFIELD
230 POKE 559,30:POKE 53277,3:POKE 54279,PMBASE
240 POKE 704,200:POKE 705,70:POKE 706,154:POKE 707,254
250 REM CLEAR OUT STRINGS
260 PO$="":PO$(256)="":PO$(2)=PO$
270 P1$="":P1$(256)="":P1$(2)=P1$
280 P2$="":P2$(256)="":P2$(2)=P2$
290 P3$="":P3$(256)="":P3$(2)=P3$
300 REM READ IN PLAYERS
310 FOR L1=0 TO 15:READ X:SO$(L1+1)=CHR$(X):NEXT L1
320 FOR L1=0 TO 15:READ X:S1$(L1+1)=CHR$(X):NEXT L1
330 FOR L1=0 TO 15:READ X:S2$(L1+1)=CHR$(X):NEXT L1
340 FOR L1=0 TO 15:READ X:S3$(L1+1)=CHR$(X):NEXT L1
350 REM EACH GETS INITIAL RANDOM VELOCITY - POSITION IS SET ON DIAGONAL
360 FOR L1=0 TO 3
370 N=2*RND(0):DX(L1)=N:IF INT(2*RND(0)) THEN DX(L1)=-DX(L1)
```

```
380 N=2*RND(0):DY(L1)=N:IF INT(2*RND(0)) THEN DY(L1)=-DY(L1)
390 X(L1)=48+30*L1:Y(L1)=32+40*L1
400 NEXT L1
410 FOR L1=0 TO 3
420 REM COLLISION TEST - IF COLLIDE REVERSE DIRECTION
430 IF PEEK(53260+L1) THEN DX(L1)=-DX(L1):DY(L1)=-DY(L1):SOUND L1,PEEK(53770),10,6
440 X(L1)=X(L1)+DX(L1):Y(L1)=Y(L1)+DY(L1)
450 REM REVERSE DIRECTION IF HITS PLAYFIELD BOUNDARY
460 IF X(L1)<48 THEN X(L1)=48:DX(L1)=ABS(DX(L1)):SOUND L1,PEEK(53770),10,6
470 IF X(L1)>183 THEN X(L1)=183:DX(L1)=-ABS(DX(L1)):SOUND L1,PEEK(53770),10,6
480 IF Y(L1)<32 THEN Y(L1)=32:DY(L1)=ABS(DY(L1)):SOUND L1,PEEK(53770),10,6
490 IF Y(L1)>207 THEN Y(L1)=207:DY(L1)=-ABS(DY(L1)):SOUND L1,PEEK(53770),10,6
500 POKE 53248+L1,X(L1):GOTO 520+L1
510 REM SHIFT STRINGS IN PM AREA
520 P0$(Y(L1))=S0$:GOTO 560
521 P1$(Y(L1))=S1$:GOTO 560
522 P2$(Y(L1))=S2$:GOTO 560
523 P3$(Y(L1))=S3$
560 SOUND L1,0,0,0:NEXT L1:POKE 53278,0:GOTO 410
1000 REM PLAYER DATA
1010 DATA 0,0,60,126,118,231,247,247,247,247,247,118,126,60,0,0
1020 DATA 0,0,60,126,102,219,251,251,247,239,223,66,126,60,0,0
1030 DATA 0,0,60,126,66,247,239,199,251,251,219,102,126,60,0,0
1040 DATA 0,0,60,126,102,231,215,215,195,247,247,118,126,60,0,0
```

While this is a clever method of obtaining fast vertical motion for players, there is no easy way to animate more than a single missile when using strings. The technique is shown more for completeness than for usefulness. We suggest that you use our player-missile subroutine for your games and avoid the complications of using string manipulation.

CHAPTER 6

VERTICAL BLANK & DISPLAY LIST INTERRUPTS

In this chapter you will learn to use Vertical Blank and Display List Interrupts. These interrupts are a powerful aid to the game programmer, who can use them to smooth animation, to enable players to be re-used in the bottom portion of the frame, to allow character sets and color registers to be changed mid-screen, and of course much more.

As you learned in the first chapter, "Vertical Blank" refers to the period of time the television set takes to reposition the electron beam back to the top left edge of the screen after raster scanning or drawing one television frame. Since television frames are generated every 1/60th of a second, there are sixty Vertical Blank periods a second, one for each frame.

Since nothing is happening on the television screen during the Vertical Blank period, it presents an ideal opportunity for a program to change the positions of objects on the screen in order to animate them. If you only knew when the Vertical Blank period occurred, you could take advantage of this time to create smooth and flicker-free animation.

The ANTIC chip in the computer lets you do just that. Since it and the GTIA/CTIA chip generate the television image, they need to know what is happening on the screen at all times. When Vertical Blank occurs, an interrupt is generated on the Atari's 6502 CPU.

An interrupt is simply a way of telling the computer to stop what it is doing, handle something more important, then return to what it was previously doing. The 6502 checks the interrupt status after executing an instruction. If an interrupt occurred, it saves the Program Counter (marking its current place in the program) and the processor status register onto the stack. It then jumps through one of a number of preset vectors, depending on the type of interrupt that occurred. Once the interrupt has been handled, the program will RTI (return from interrupt instruction) back to the instruction that was to be executed before the interruption.

ANTIC also generates Display List Interrupts. If you recall the discussion on display lists, you remember that the display list is really a program for ANTIC. Each byte in the display list is part of a series of instructions that lets ANTIC know what type of information is to be displayed. Setting the high bit in an instruction in the display list will trigger an interrupt when it is time for the television beam to display the information for the last scan line in that graphics mode line. This type of interrupt is sometimes called a "Raster Interrupt." Remember that the interrupt does not stop the TV raster process but causes the 6502 to execute a series of instructions at this specific time.

6 VERTICAL BLANK AND DISPLAY LIST INTERRUPTS

Vertical Blank and Display List Interrupts (DLI's) are sent to the 6502 Non-Maskable Interrupt (NMI) vector at $FFFA. It is called non-maskable because these interrupts cannot be disabled on the 6502 CPU as can other types of interrupts. Three interrupts go to the NMI vector. They are Vertical Blank, DLI's, and System Reset. Although these interrupts cannot be masked off on the 6502, ANTIC filters the first two. A memory-mapped hardware register in ANTIC, NMIEN ($D40E), allows the programmer to enable or disable Vertical Blank and Display List Interrupts. Another register, NMIST ($D40F), shows which interrupt actually occurred. When an NMI occurs, the 6502 goes to an OS routine to find out what caused the interrupt. If a DLI occurred, this routine vectors through page-two vectors to a DLI service routine that changes one or more graphics registers which control display. On the other hand, if a VBI occurred, the OS routine saves the Accumulator, X and Y registers on the stack, then jumps through the appropriate page-two vector to the Vertical Blank routine. And if System Reset was pressed, it goes directly to the OS warm start vector for system reset ($E474).

Vertical Blank Interrupts

The Atari computers use the Vertical Blank Interrupt for many housekeeping chores. When the Atari is first powered up under normal conditions, the Vertical Blank Interrupt is enabled. A special list of vectors is placed in page two of memory. The operating system places the list of vectors in page two because these are RAM locations and the user can change their contents to point to his own interrupt service routine. System Reset is the only interrupt that cannot be routed through page two.

The operating system's Vertical Blank Interrupt (VBI) service routine is a two-stage process. When an interrupt occurs, the computer is sent to a vector in the operating system, which in turn sends it to a routine to determine what type of interrupt has occurred. It then jumps through the appropriate vector on page two. It is called two-stage because it goes through two vectors on page two. This allows the programmer to use either his own VBI routine or that of the OS, or both.

The OS uses the VBI routine to transfer data from some of the special hardware registers in Atari's custom chips to shadowed locations in RAM and vice versa. For example, it reads the color register information, the location of the display list, and other graphics control information placed in RAM by the user, and writes them to their hardware addresses. It also reads many hardware registers such as game controllers and light pen and places them in RAM for the user. It updates the clock at locations 18-20 ($12-$14), and updates the timer for the attract mode which cycles the colors if a key has not been pressed for several minutes. It even takes care of the repeat action on the keyboard keys. Because the VBI routine updates many registers that are used by the average program, most programmers don't wish to replace it entirely by their own routine, since in many cases it would mean duplicating at least some of the procedures that are handled automatically by the OS.

The vectors are called Immediate VBlank and Deferred VBlank. Atari engineers split the Vertical Blank period into two separate sections because a VBI can extend for about 20,000 cycles or nearly all the time available before the next VBI. The

Immediate VBlank portion refers to the time critical period when the electron beam is actually offscreen. It is here that shadowing and hardware registers are updated. Once the OS VBI routine has completed its housekeeping chores, it jumps through the Deferred VBlank vector, which is set by the OS to point to a routine that will restore the registers and return the computer to its position before the interrupt. The OS VBI routine will abort if the computer was in the middle of a time critical I/O routine such as sending data to a cassette or disk drive. In such a case, the Deferred VBlank vector is bypassed. Unless you are using disk I/O where your code must be short enough not to delay shadowing updating, you can use Deferred VBlank without being concerned.

A major question asked by many programmers is: How much time do I have in the VBI routine to execute my code? To answer this we must examine the TV process again.

A television image is composed of 525 lines. Because of what is called interlacing, only half are drawn per frame, and some of these are offscreen due to normal vertical overscan. It takes the TV 63.5 microseconds to draw a single line. This includes the time it takes to shut the electron beam off and reposition it back on the left edge one scan line below. The length of Vertical Blank is equivalent to 22 scan lines or roughly 1400 microseconds. With the Atari's 6502B CPU running at 1.79 MHz, 1400 microseconds is equivalent to approximately 2500 machine cycles (1400 X 1.79 = 2506). So, how much can you do "inside" the Vertical Blank period? Not very much, if you are trying to accomplish things like moving players and scrolling the screen solely inside the Vertical Blank period. The important thing to remember is that one television frame is 1/60th of a second, which is equivalent to 16,666 microseconds or just about 30,000 machine cycles. This means that there is a maximum of approximately 30,000 machine cycles BETWEEN Vertical Blanks. ANTIC delays processor time in order to fetch screen data and look up character data during the screen drawing process. A programmer using an extensive Vertical Blank Interrupt routine will want to be sure that the routine is finished before the next VBI occurs; otherwise, his routine will be aborted in the middle unless very special precautions are taken. A programmer will also want to be sure that his VBlank routine is not so long that it causes serious delays to the program code that is running outside VBlank.

Programmers often ask if the 2500 cycle VBlank time constraint seriously limits the amount of time for smooth offscreen animation and updating. The answer is No! True, all of the graphics updating must be performed while the beam is offscreen, but you can also gain some additional time. Remember what a normal display list looks like. It begins with 24 blank lines. That gives 24 by 63.5 microseconds/line or another 2728 cycles that can be considered offscreen. High scores, player scores and messages generally cover the top few lines too. This adds additional time offscreen. Obviously if your Vertical Blank routine is even longer, you should be OK as long as you do all of your graphics updating within the first 5200 cycles of your routine. Figure that you have a maximum of 20,000 cycles (4500 instructions) in Deferred VBlank, but you must finish before the next VBI or your program will crash.

While programmers have written entire games in Deferred VBlank, only certain operations should be put in VBlank. All graphics updating, including scrolling the screen, moving player-missile objects, and changing color registers, should defi-

nitely be done in VBlank. In addition, collisions should be checked, and joysticks or paddles should be read. This is also the best place to implement time critical sound routines. Everything else, including calculations, should be in your main code outside VBlank.

Setting up a Vertical Blank Interrupt is a simple process since the OS has a routine to set up the vector for you. All it involves is storing the address of your VBI routine in the vector table on page two of RAM. If a VBI has occurred between the time the two bytes that make up the vector were stored in the table, the 6502 would jump through an erroneous address, and the program would become erroneously lost. The OS setup routine automatically assures this never happens.

Setting up the routine is simple. Load the Accumulator with 7 if you are setting a Deferred VBlank routine, and 6 if you are setting an Immediate VBlank routine. The X register is loaded with the high order byte of your routine, the Y register with the low order byte. You then JSR to the OS SETVBV routine at $E45C.

Example:

```
SETVBV   .EQ $E45C
         LDA #$07     ;DEFERRED
         LDX /VBLANK ;HIGH BYTE OF USER ROUTINE
         LDY #VBLANK ;LOW BYTE
         JSR SETVBV
```

There are two possible exit points from a VBlank routine depending on whether Immediate or Deferred VBlank is used. If the programmer uses the Immediate one and still desires to use the OS VBI routine, the vector is $E45F (SYSVBV). If the Deferred VBlank is used or the programmer does not want the OS VBI routine to execute the vector, then it is $E462 (XITVBV). The XITVBV routine pulls the registers off the stack and does a RTI (Return from Interrupt).

Display List Interrupts

Display list interrupts are generally used to change the color registers mid-screen or switch character sets in use. These changes can be made very rapidly since they are short and usually modify only a few bytes. Take care so that changes of this type are offscreen during Horizontal Blank, or they will appear crude and annoying. When ANTIC encounters the DLI instruction, it completes the last scan line for the mode it is drawing, then services the interrupt. This means in effect that the interrupt must be set during the mode line above the one you want the interrupt to effect.

The period of Horizontal Blank is seven microseconds. Horizontal Overscan is another 3.31 microseconds. These 10.31 microseconds mean approximately eighteen machine cycles offscreen. In order for an interrupt routine to synchronize with Horizontal Blank, a special hardware register in ANTIC freezes the 6502 until Horizontal Blank occurs. This register at $D40A is known as WSYNC. Writing to this location pulls down the ready line on the CPU until Horizontal Sync. If you insert a STA WSYNC instruction, then change the value in a color register, color

won't be changed in the middle of the current line but will go into effect when the beam is off the left edge of the screen one scan line lower.

What if eighteen cycles are not enough time to make all the changes you need? There are several approaches that can be taken, and the proper one depends of course upon the situation.

As with the VBI, you need not worry about crashing the program because the code does not fit in the Horizontal Blank time or even within the time it takes to do the entire scan line. Only if the code is so long that another DLI occurs before the previous one has finished would you be likely to run into problems and crash the program. There is no reason to believe that you must complete the interrupt routine by the end of the scan line. There are some programs that have one DLI set at the top of the screen and do not return from the interrupt until the entire screen has been drawn, hundreds of scan lines and thousands of machine cycles later. The interrupt routine is used to control graphics information to the screen, line by line. A DLI routine written in this manner is called a kernel.

Programmers using DLI's for simple screen changes should decide if all of them must be made on a single line. Perhaps only a few changes need be made immediately. The rest can be made on the next line by waiting again for Horizontal Sync. All the changes must be made on a single line, if there is a major screen division between the score line and the playfield requiring different colors and character set. In this case, it might be practical to insert a zero in the display list. A zero is the blank line instruction in a display list. Changes could be made past Horizontal Blank on this line while the raster beam is drawing it and still be invisible to the viewer.

There is one more important point for those readers who still need to count cycles. Although a horizontal line takes 63.5 microseconds, you do not have 113 cycles per line (63.5 x 1.79 = 113.6). This is because ANTIC's Direct Memory Access (DMA) ability allows it to freeze the 6502 CPU and steal cycles in order to get screen data from screen memory, player/missile data from player/missile memory, and to look up character set data. It even steals cycles to look at the display list so it knows what type of information to display. The amount of time stolen per scan line can vary. ANTIC steals a cycle for each byte in memory it must access. This DMA is controlled by a hardware register called DMACTL at $D400 (54272 decimal). The OS VBlank routine rewrites the value found at its shadow (SDMCTL) at $22F (559 decimal) every VBI. By setting bits in SDMCTL, the program can control screen width from either 128, 160, or 228 color clocks. Selecting screen width tells ANTIC it may steal cycles from the 6502 to access the display list and screen memory in order to display information. Bits are also set here so that ANTIC may steal cycles to look up player and/or missile data in memory.

If you want an idea of how much time you lose from DMA, try the following example in BASIC.

```
10 FOR L1=1 TO 1000:NEXT L1
```

Then try:

```
10 POKE 559,0:FOR L1=1 TO 1000:NEXT L1:POKE 559,32
```

The second example is approximately 30% faster. The first POKE disables ANTIC's DMA, and the screen becomes black. The second POKE restores DMA after the loop is completed so we know how long it takes. If you were to try the first example in different graphic modes, you would find Graphics 8 the worst case. This is because Graphics 8 uses much more memory, so ANTIC must steal more cycles in order to access more screen memory. However, the Graphics 0 text mode isn't much faster because ANTIC also steals cycles while retrieving the characters set.

Display List Interrupts are even simpler to set up than Vertical Blank Interrupts. Since there is no DLI enabled when the program takes control, the program simply has to store the low byte-high byte address of the routine into the vector on page two (VDSLST—$200,$201). Once the vector has been stored and the display list is set for an interrupt, NMIEN is set to enable DLI's. Bit 7 of NMIEN enables DLI's. Bit 6 enables VBI's. Storing a $C0 (192 decimal) is all that is needed to enable the DLI.

Example:

```
VDSLST   .EQ $200     ;DISPLAY LIST INTERRUPT VECTOR
NMIEN    .EQ $D40E
         LDA #VBI     ;LOW BYTE OF DLI ROUTINE
         STA VDSLST
         LDA /VBI     ;HIGH BYTE OF DLI ROUTINE
         STA VDLIST+1
         LDA #$C0
         STA NMIEN    ;ENABLE DLI'S
*                     ;WITHOUT DISABLING VBI'S
```

Since the OS does not save the Accumulator, X and Y registers for a DLI as it does for a VBI, the user must save any registers used in the DLI routine on the stack upon entering the interrupt routine and restore them before exiting. If this is not done, the program will unquestionably crash.

The power of DLI's is obvious. With a DLI, a program can easily switch character bases from the standard ROM text set to a redefined character graphics set. Without a DLI, the programmer could only redefine characters that would not be used in the text display. DLI's enable you to change color registers mid-screen. A good example is a game like *Sea Wolf* (a submarine game) in which the background color changes from sky blue to ocean blue partway down the screen. Without a DLI to change the background color, one other playfield color register would have to be used for either the sky or ocean, thus limiting further the amount of color available for the screen. Multiple DLI's can be used to control screen scrolling in games like *Frogger* in which different bands on the screen scroll in different directions at different speeds.

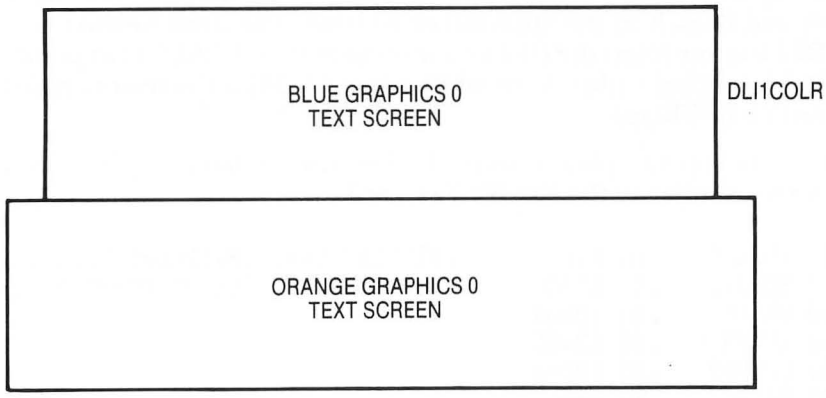
DLI's can change player/missile horizontal positions and colors so players can be reused down the screen as long as vertical positions do not need to overlap in the same player. A player, which is actually a long vertical stripe, can have several different smaller images. The DLI allows you to snip the strip apart and reposition those pieces in the lower portion by changing the player's horizontal position register. If vertical movement is required, you must be careful that the different images don't cross the boundary. Since collision registers are set by the hardware when collision occurs, collision registers can be read within the interrupt routine so

that hardware collision detection is not lost by reusing the players. A good example of reusing the players is the Atari *Galaxian* cartridge.

BASIC can use a DLI to change one or more color registers at a particular spot mid-screen, if a short Machine language routine is added. The example below changes the background color register in the text mode to orange and darkens the color of the characters so they show up against the background. This is controlled by the color in playfield 2. The change occurs in line 12 so that the interrupt is set in the mode line preceding the change. The modified display list is as follows.

```
112
112      Blank 24 scan lines
112
 66 )    LMS for BASIC 0 (ANTIC 2) (64+2)
 64 }    Low byte of screen memory
156 )    High byte of screen memory
  2      second mode line
  2
  2
  2
  2
  2
  2
  2
  2
130      Eleventh mode line + DLI (128+2)
  2
  2
  .
  .
```

Since the display List is virtually identical, except for the modification in the eleventh mode line, we need only change that byte to activate our DLI. This is at the DLIST+15 byte.



```
5 REM FIND DISPLAY LIST
10 DLIST=PEEK(560)+256*PEEK(561)
15 REM INSERT INTERRUPT INSTRUCTION
20 POKE DLIST+15,130
25 REM READ IN DLI SERVICE ROUTINE
30 FOR I=0 TO 19
40 READ A:POKE 1536+I,A:NEXT I
50 REM POKE IN INTERRUPT VECTOR
60 POKE 512,0:POKE 513,6
70 REM ENABLE DLI
80 POKE 54286,192
90 DATA 72,138,72,169,38,162,90
100 DATA 141,10,212,141,26,208
110 DATA 141,24,208,104,170,104,64
```

The actual Machine language DLI service routine is as follows:

```
D01A:             00010 COLBK    .EQ $D01A
D018:             00020 COLPF2   .EQ $D018
D40A:             00030 WSYNC    .EQ $D40A
4000: 48          00040          PHA          ;SAVE ACCUMULATOR
4001: 8A          00050          TXA
4002: 48          00060          PHA          ;SAVE X-REGISTER
4003: A9 52       00070          LDA #$52     ;DARK COLOR FOR CHARACTERS
4005: A2 26       00080          LDX #$26     ;ORANGE BACKGROUND
4007: 8D 0A D4    00090          STA WSYNC    ;WAIT
400A: 8D 1A D0    00100          STA COLBK    ;STORE COLOR
400D: 8D 18 D0    00110          STA COLPF2   ;STORE COLOR
4010: 68          00120          PLA
4011: AA          00130          TAX
4012: 68          00140          PLA
4013: 40          00150          RTI
```

Care must be taken when using multiple DLI's. There is only one vector for a DLI. Setting NMIEN enables DLI's immediately and therefore does not wait for the next frame. Your program must insure that the proper routine will be executed. Three methods are available. The first method is to use a variable as an index that is incremented by each DLI. The program then branches to the appropriate routine depending on the value of the index. The second method is to read the vertical line counter and branch to the appropriate routine. The third method is the cleanest. Each DLI routine resets the DLI vector on page two (VDSLST) to point to the next. The DLI is enabled within Vertical Blank and VDSLST is reset to point to the first DLI routine in VBlank.

The following example is a simple DLI routine to change the background color of a text screen similar to the *Sea Wolf* example.

```
00010 VDSLST   .EQ $200      ;DISPLAY LIST INTERRUPT VECTOR
00020 SDLSTL   .EQ $230      ;STARTING ADDRESS OF DISPLAY LIST
00030 WSYNC    .EQ $D40A
00040 NMIEN    .EQ $D40E
00050 COLPF2   .EQ $D018
00060 PAGE0    .EQ $F0
```

```
00070           LDA SDLSTL     ;FIND DISPLAY LSIT
00080           STA PAGE0      ;DISPLAY LIST LOW BYTE
00090           LDA SDLSTL+1
00100           STA PAGE0+1    ;SAVE HIGH BYTE
00110           LDY #$08       ;3RD TEXT LINE INA A GRAPHICS 0 DISPLAY LIST
00120           LDA (PAGE0),Y ;GET DISPLAY LSIT INSTRUCTION
00130           ORA #%10000000;TURN ON HIGH BIT ($80)
00140           STA (PAGE0),Y ;AND STORE IT BACK IN THE DISPLAY LIST
00150           LDA #DLI       ;DLI LOW BYTE
00160           STA VDSLST     ;STORE LOW BYTE OF OUR ROUTINE IN DLI VECTOR
00170           LDA /VBI       ;GET HIGH BYTE OF OUR DLI ROUTINE
00180           STA VDSLST+1   ;STORE HIGH BYTE OF VECTOR
00190           LDA #%11000000; ENABLE DLI AND KEEP VBI ($C0)
00200           STA NMIEN
00210           RTS            ;AN RTS SHOULD RETURN THE PROGRAM
00220 *                        ;BACK TO THE DEBUGGER WHEN RUN
00230 DLI       PHA            ;SAVE ANY REGISTER THAT WILL BE USED
00240           LDA #$A2       ;DEEP BLUE
00250           STA WSYNC      ;WAIT FOR SYNC FOR NEXT LINE
00260           STA COLPF2
00270           PLA            ;RESTORE ACCUMULATOR
00280           RTI            ;RETURN TO MAIN PROGRAM
```

Binary representation was used for clarity in this example.

Kernels

A kernel is a special DLI routine designed to control graphics information on a line-by-line basis for the entire screen. It does this by monitoring the VCOUNT (vertical line counter) register. The most frequent use is for producing multi-colored players. Even the player width can be changed on a line-by-line basis. For example, a cowboy image could be made out of one player; a broad white hat defined at double width changes a few scan lines down to a slim pink face, then changes a few scan lines down to a brown cowboy suit and finally to black boots; four colors and two different resolutions in a single player. Another example is the players in Atari's *Basketball* cartridge. You cannot create multi-colored players like these without Display List Interrupts because DLI's are keyed to playfield vertical positions, not player positions.

Kernels are difficult to use because graphics information changes only during Horizontal Blank. Kernels also drastically reduce the amount of time available for program logic, since most of the 6502's time is spent writing graphics information and waiting for Horizontal Sync on the next line. Since virtually no computation time is available during display time, and only about 4000 cycles are available during Vertical Blank and overscan time, kernels are limited to simple skill and action games.

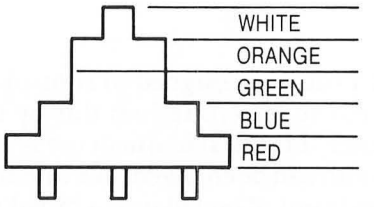
Multi-Colored Joystick Controlled Player

The following example illustrates the use of a kernel to produce a joystick-movable, multi-colored player. The player is 15 bytes high, but it also includes two zero bytes on each side so that it erases itself as it moves. There is no actual erase feature in this example. The player is controlled by a joystick. This is not the standard routine but an alternate one that uses table lookup to determine its horizontal and vertical movement. The formulas are as follows:

PLAYRH = PLAYRH + HOFF(STICK)
PLAYRV = PLAYRV + VOFF(STICK)

For example, if the joystick points to the right and up, STICK has the value 6. HOFF(6)=$02, and VOFF(6)=$FE or a−2. The player position changes to a location two scan lines up and two pixels right.

The kernel itself is designed to change the player's color four different times. To accomplish this it must wait for the vertical line counter to reach the player's position. Since this counter, called VCOUNT, actually increments every other scan line, we divide the player's vertical position by two. We add one because we actually want to start one line later. We have to wait for each horizontal scan to test for whether VCOUNT has reached our player's position. If it has, we then begin fetching color to put in the player's color register. Since the color changes every three scan lines, we must wait for three more horizontal scan lines before changing colors. The fact that we have to do a WSYNC for each scan line prevents us from doing any calculations ouside of Vertical Blank.




```
                00010 *KERNEL EXAMPLE - MULTICOLORED PLAYER -MOVES WITH JOYSTICK
                00020 *CODED BY DAN PINAL
6000:           00040 PDATA     .EQ $6000
D407:           00050 PMBASE    .EQ $D407
0230:           00060 SDLSTL    .EQ $230
D000:           00070 HPOSP0    .EQ $D000
02C0:           00080 PCOLR0    .EQ $2C0
D012:           00090 COLPM0    .EQ $D012
D40A:           00100 WSYNC     .EQ $D40A
D40B:           00110 VCOUNT    .EQ $D40B
D20E:           00120 IRQEN     .EQ $D20E
D40E:           00130 NMIEN     .EQ $D40E
0200:           00140 VDSLST    .EQ $200
0224:           00150 VVBLKD    .EQ $224
022F:           00160 SDMCTL    .EQ $22F
```

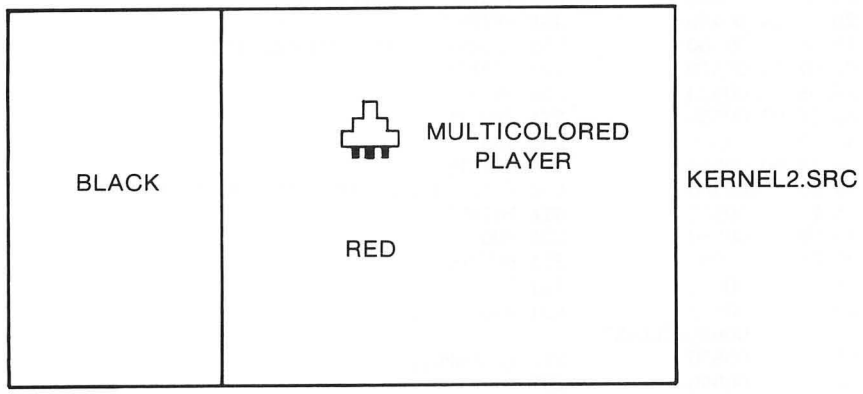
```
D400:           00170 DMACTL   .EQ $D400
D01D:           00180 GRACTL   .EQ $D01D
E45C:           00190 SETVBV   .EQ $E45C
E462:           00200 XITVBV   .EQ $E462
6300:           00210 MISSL0   .EQ PDATA+$300
6400:           00220 PLAYR0   .EQ MISSL0+$100
6500:           00230 PLAYR1   .EQ PLAYR0+$100
6600:           00240 PLAYR2   .EQ PLAYR1+$100
6700:           00250 PLAYR3   .EQ PLAYR2+$100
0278:           00260 STICK0   .EQ $278
                00270 *ZERO PAGE EQUATES
00F0:           00280 VTEMP0   .EQ $F0
00F1:           00290 VTEMP1   .EQ $F1
00F8:           00300 POTMP0   .EQ $F8
00F9:           00310 POTMP1   .EQ $F9
                00340 *
                00350 *
                00360 INIT
4000: A9 F9     00370          LDA #DLIST
4002: 8D 30 02  00380          STA SDLSTL
4005: A9 40     00390          LDA /DLIST
4007: 8D 31 02  00400          STA SDLSTL+1
400A: A9 98     00410          LDA #KERNEL ; SET DISPLAY LIST INTERRUPT
400C: 8D 00 02  00420          STA VDSLST
400F: A9 40     00430          LDA /KERNEL
4011: 8D 01 02  00440          STA VDSLST+1
4014: A0 56     00450          LDY #VBLANK ; SET VERTICAL BLANK
4016: A2 40     00460          LDX /VBLANK
4018: A9 07     00470          LDA #$07    ;DEFFERRED
401A: 20 5C E4  00480          JSR SETVBV
401D: A9 60     00490          LDA /PDATA  ; INIT PM GRAPHICS
401F: 8D 07 D4  00500          STA PMBASE
4022: A9 3E     00510          LDA #$3E
4024: 8D 2F 02  00520          STA SDMCTL
4027: A9 03     00530          LDA #03
4029: 8D 1D D0  00540          STA GRACTL
402C: A9 63     00550          LDA /MISSL0 ; CLEAR PLAYER/MISSILE RAM
402E: 85 F9     00560          STA POTMP1
4030: A9 00     00570          LDA #00
4032: 85 F8     00580          STA POTMP0
4034: A8        00590          TAY
4035: A2 05     00600          LDX #$05    ; 5 PAGES
                00610 CLEARP
4037: 91 F8     00620          STA (POTMP0),Y
4039: C8        00630          INY
403A: D0 FB     00640          BNE CLEARP
403C: E6 F9     00650          INC POTMP1
403E: CA        00660          DEX
403F: D0 F6     00670          BNE CLEARP
4041: A9 0E     00680          LDA #$0E    ; WHITE
4043: 8D C0 02  00690          STA PCOLR0
4046: A9 80     00700          LDA #$80
4048: 8D FD 40  00710          STA PLAYRH
404B: 8D FE 40  00720          STA PLAYRV
                00730 ; ENABLE INTERRUPTS
404E: A9 C0     00740          LDA #$C0
4050: 8D 0E D4  00750          STA NMIEN
                00760 HERE
4053: 4C 53 40  00770          JMP HERE    ;ENDLESS LOOP OUTSIDE VBLANK
                00780 *VBLANK ROUTINE
                00790 VBLANK
4056: A9 C0     00800          LDA #$C0
```

```
4058: 8D 0E D4 00810          STA NMIEN   ; REENABLE DLI (FOR REV. A O.S.)
405B: AE 78 02 00820          LDX STICKO  ; JOYSTICKO VALUE
405E: 18       00830          CLC
405F: AD FD 40 00840          LDA PLAYRH  ; LOAD OLD HORIZ. PLAYER POS
4062: 7D D5 40 00850          ADC HOFF,X  ; ADD HORIZ DIRECTION VECTOR
4065: C9 30    00860          CMP #$30
               00870 * CHECK IF PAST RIGHT OR LEFT EDGE
4067: 90 0A    00880          BCC NEWV
4069: C9 D0    00890          CMP #$D0
406B: B0 06    00900          BCS NEWV
406D: 8D FD 40 00910          STA PLAYRH  ;STORE NEW HORIZ PLAYER POSITION
4070: 8D 00 D0 00920          STA HPOSP0  ;TELL ANTIC
               00930 NEWV
4073: 18       00940          CLC
4074: AD FE 40 00950          LDA PLAYRV  ;LOAD OLD VERT PLAYER POS
4077: 7D E5 40 00960          ADC VOFF,X  ;ADD VERT DIRECTION VECTOR
               00970 * CHECK IF PAST TOP OR BOTTOM
407A: C9 22    00980          CMP #$22
407C: 90 17    00990          BCC XVBLANK
407E: C9 D0    01000          CMP #$D0
4080: B0 13    01010          BCS XVBLANK
4082: 8D FE 40 01020          STA PLAYRV  ;STORE NEW VERT POSITION
4085: 85 F0    01030          STA VTEMP0  ;LO BYTE OF P/M AREA
4087: A9 64    01040          LDA /PLAYR0
4089: 85 F1    01050          STA VTEMP1  ;HI BYTE OF P/M AREA
408B: A0 13    01060          LDY #$13    ;20 ELEMENTS
               01070 PDRAW
408D: B9 C2 40 01080          LDA IMAGE,Y ;GET BYTE FROM SHAPE TABLE
4090: 91 F0    01090          STA (VTEMP0),Y ;STORE BYTE IN P/M AREA
4092: 88       01100          DEY         ;NEXT BYTE
4093: 10 F8    01110          BPL PDRAW   ;DONE?
               01120 XVBLANK
4095: 4C 62 E4 01130          JMP XITVBV
               01140 *
               01150 KERNEL
4098: 48       01160          PHA         ; SAVE REGISTERS
4099: 8A       01170          TXA
409A: 48       01180          PHA
409B: 98       01190          TYA
409C: 48       01200          PHA
409D: A2 03    01210          LDX #$03    ; USE X FOR INDEX TO COLOR TABLE
409F: AD FE 40 01220          LDA PLAYRV  ; GET VERT. POS.
40A2: 4A       01230          LSR         ; VCOUNT COUNTS EVERY OTHER LINE
40A3: A8       01240          TAY
40A4: C8       01250          INY
               01260 UNTIL
40A5: 8D 0A D4 01270          STA WSYNC   ;WAIT TILL OFF SCREEN
40A8: CC 0B D4 01280          CPY VCOUNT  ;ARE WE AT LINE TO BEGIN CHANGING (
40AB: B0 F8    01290          BCS UNTIL   ;NO HAVEN'T REACHED IT-BRANCH
               01300 CHANGE
40AD: BD F5 40 01310          LDA COLOR,X ; GET COLOR
40B0: 8D 12 D0 01320          STA COLPM0  ; STUFF COLOR
40B3: 8D 0A D4 01330          STA WSYNC   ; WAIT 3 LINES # DO NEXT CHANGE
40B6: 8D 0A D4 01340          STA WSYNC
40B9: 8D 0A D4 01350          STA WSYNC
40BC: CA       01360          DEX
40BD: 10 EE    01370          BPL CHANGE  ; GO BACK TILL ALL CHANGES MADE
40BF: 4C 62 E4 01380          JMP XITVBV  ; LET O.S. RESTORE REGISTERS
               01390 *
40C2: 00 00 10
40C5: 10 10 10
```

```
40C8: 38 38    01400 IMAGE    .HS 0000101010103838
40CA: 38 38 7C
40CD: 7C 7C FE
40D0: FE 54    01410          .HS 38387C7C7CFEFE54
40D2: 54 00 00 01420          .HS 540000
40D5: 00 00 00
40D8: 00 00 02
40DB: 02 02    01430 HOFF     .HS 0000000000020202
40DD: 00 FE FE
40E0: FE 00 00
40E3: 00 00    01440          .HS 00FEFEFE00000000
40E5: 00 00 00
40E8: 00 00 02
40EB: FE 00    01450 VOFF     .HS 0000000000002FE00
40ED: 00 02 FE
40F0: 00 00 02
40F3: FE 00    01460          .HS 0002FE000002FE00
40F5: 44 76 B8
40F8: 3A       01470 COLOR    .HS 4476B83A
40F9: 80 41    01480 DLIST    .HS 8041
40FB: F9 40    01490          .DA DLIST
40FD: 00       01500 PLAYRH   .HS 00
40FE: 00       01510 PLAYRV   .HS 00
```

Splitting Screen Horizontally Using DLIs

Our second example of a kernel produces a split color screen, but this time the split is across the vertical axis with the left portion of the screen black and the right pink. The kernel takes advantage of a careful timing loop after a WSYNC instruction. It waits until the beam is offscreen for each scan line, so the beam always reaches the same horizontal position before changing the background color register to pink. The loop is a simple countdown from six to zero. The background is restored to black for the next scan line immediately after the WSYNC. Notice that the color change is fed directly to the hardware color register. Remember that the kernel operates outside of VBlank, and the shadowed background color register doesn't get copied until VBlank. Thus, color changes must feed directly to hardware. We do have a joystick-controlled player-missile routine operating in VBlank in order to show that the kernel and it are entirely independent.



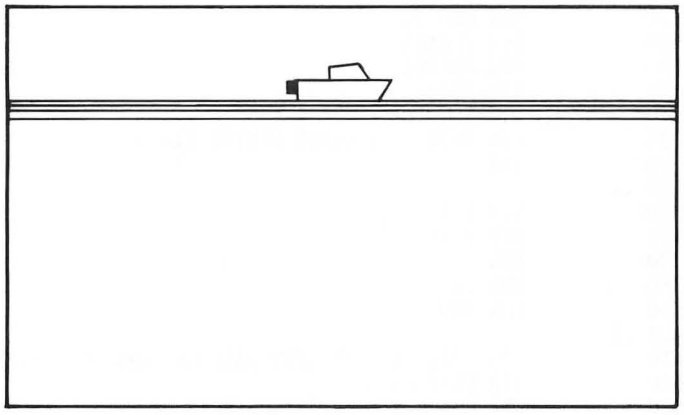
```
                00010 *KERNEL EXAMPLE - BY DAN PINAL
                00020 *        KERNEL CHANGES COLOR MIDSCREEN DURING EACH SCAN LINE
6000:           00040 PDATA    .EQ $6000
D407:           00050 PMBASE   .EQ $D407
0230:           00060 SDLSTL   .EQ $230
D000:           00070 HPOSPO   .EQ $D000
02C0:           00080 PCOLR0   .EQ $2C0
D012:           00090 COLPM0   .EQ $D012
D01A:           00100 COLBAK   .EQ $D01A
D40A:           00110 WSYNC    .EQ $D40A
D40B:           00120 VCOUNT   .EQ $D40B
D20E:           00130 IRQEN    .EQ $D20E
D40E:           00140 NMIEN    .EQ $D40E
0200:           00150 VDSLST   .EQ $200
0224:           00160 VVBLKD   .EQ $224
022F:           00170 SDMCTL   .EQ $22F
D400:           00180 DMACTL   .EQ $D400
D01D:           00190 GRACTL   .EQ $D01D
E45C:           00200 SETVBV   .EQ $E45C
E462:           00210 XITVBV   .EQ $E462
6300:           00220 MISSL0   .EQ PDATA+$300
6400:           00230 PLAYR0   .EQ MISSL0+$100
6500:           00240 PLAYR1   .EQ PLAYR0+$100
6600:           00250 PLAYR2   .EQ PLAYR1+$100
6700:           00260 PLAYR3   .EQ PLAYR2+$100
0278:           00270 STICK0   .EQ $278
                00280 *ZERO PAGE EQUATES
00F0:           00290 VTEMP0   .EQ $F0
00F1:           00300 VTEMP1   .EQ $F1
00F8:           00310 POTMP0   .EQ $F8
00F9:           00320 POTMP1   .EQ $F9
                00350 *
                00360 *
                00370 INIT
4000: A9 FB     00380          LDA #DLIST
4002: 8D 30 02  00390          STA SDLSTL
4005: A9 40     00400          LDA /DLIST
4007: 8D 31 02  00410          STA SDLSTL+1
400A: A9 98     00420          LDA #KERNEL ; SET DISPLAY LIST INTERRUPT
400C: 8D 00 02  00430          STA VDSLST
400F: A9 40     00440          LDA /KERNEL
4011: 8D 01 02  00450          STA VDSLST+1
4014: A0 56     00460          LDY #VBLANK ; SET VERTICAL BLANK
4016: A2 40     00470          LDX /VBLANK
4018: A9 07     00480          LDA #$07    ;DEFFERRED
401A: 20 5C E4  00490          JSR SETVBV
401D: A9 60     00500          LDA /PDATA  ; INIT PM GRAPHICS
401F: 8D 07 D4  00510          STA PMBASE
4022: A9 3E     00520          LDA #$3E
4024: 8D 2F 02  00530          STA SDMCTL
4027: A9 03     00540          LDA #03
4029: 8D 1D D0  00550          STA GRACTL
402C: A9 63     00560          LDA /MISSL0 ; CLEAR PLAYER/MISSILE RAM
402E: 85 F9     00570          STA POTMP1
4030: A9 00     00580          LDA #00
4032: 85 F8     00590          STA POTMP0
4034: A8        00600          TAY
4035: A2 05     00610          LDX #$05    ; 5 PAGES
                00620 CLEARP
4037: 91 F8     00630          STA (POTMP0),Y
4039: C8        00640          INY
```

```
403A: D0 FB     00650           BNE CLEARP
403C: E6 F9     00660           INC POTMP1
403E: CA        00670           DEX
403F: D0 F6     00680           BNE CLEARP
4041: A9 0E     00690           LDA #$0E    ; WHITE
4043: 8D C0 02  00700           STA PCOLR0
4046: A9 80     00710           LDA #$80
4048: 8D FF 40  00720           STA PLAYRH
404B: 8D 00 41  00730           STA PLAYRV
                00740 * ENABLE INTERRUPTS
404E: A9 C0     00750           LDA #$C0
4050: 8D 0E D4  00760           STA NMIEN
                00770 HERE
4053: 4C 53 40  00780           JMP HERE
                00790 *
                00800 VBLANK
4056: A9 C0     00810           LDA #$C0
4058: 8D 0E D4  00820           STA NMIEN   ; REENABLE DLI (FOR REV. A O.S.)
405B: AE 78 02  00830           LDX STICK0
405E: 18        00840           CLC
405F: AD FF 40  00850           LDA PLAYRH
4062: 7D D7 40  00860           ADC HOFF,X
4065: C9 30     00870           CMP #$30
                00880 * CHECK IF PAST RIGHT OR LEFT EDGE
4067: 90 0A     00890           BCC NEWV
4069: C9 D0     00900           CMP #$D0
406B: B0 06     00910           BCS NEWV
406D: 8D FF 40  00920           STA PLAYRH
4070: 8D 00 D0  00930           STA HPOSP0  ; TELL ANTIC
                00940 NEWV
4073: 18        00950           CLC
4074: AD 00 41  00960           LDA PLAYRV
4077: 7D E7 40  00970           ADC VOFF,X
                00980 * CHECK IF PAST TOP OR BOTTOM
407A: C9 22     00990           CMP #$22
407C: 90 17     01000           BCC XVBLANK
407E: C9 D0     01010           CMP #$D0
4080: B0 13     01020           BCS XVBLANK
4082: 8D 00 41  01030           STA PLAYRV
4085: 85 F0     01040           STA VTEMP0
4087: A9 64     01050           LDA /PLAYR0
4089: 85 F1     01060           STA VTEMP1
408B: A0 13     01070           LDY #$13    ; 20 ELEMENTS
                01080 PDRAW
408D: B9 C4 40  01090           LDA IMAGE,Y
4090: 91 F0     01100           STA (VTEMP0),Y
4092: 88        01110           DEY
4093: 10 F8     01120           BPL PDRAW
                01130 XVBLANK
4095: 4C 62 E4  01140           JMP XITVBV
                01150 ;
                01160 KERNEL
4098: 48        01170           PHA         ; SAVE REGISTERS
4099: 8A        01180           TXA
409A: 48        01190           PHA
409B: 98        01200           TYA
409C: 48        01210           PHA
409D: A0 70     01220           LDY #$70
                01230 DALOOP
409F: 8D 0A D4  01240           STA WSYNC   ; WAIT TILL BEAM OFFSCREEN
40A2: A9 00     01250           LDA #$00    ; BLACK
```

```
40A4: A2 06     01260          LDX #$06     ; DELAY TIMER
                01270 UNTIL
40A6: CC 0B D4 01280           CPY VCOUNT   ; STILL ON SCREEN?
40A9: 90 0E     01290          BCC XKERNEL  ; LEAVE IF TOO LOW
40AB: 8D 1A D0 01300           STA COLBAK   ; TURN BACKGROND TO BLACK
                01310 HANGON
40AE: CA        01320          DEX          ; COUNTDOWN FROM 6 TO 0
40AF: D0 FD     01330          BNE HANGON   ; BEAM MID-LINE YET
40B1: A2 44     01340          LDX #$44     ; PINK
40B3: 8E 1A D0 01350           STX COLBAK   ; TURN SCREEN PINK
40B6: 4C 9F 40 01360           JMP DALOOP   ; GO GET READY FOR NEXT LINE
                01370 XKERNEL
40B9: A9 00     01380          LDA #$00     ; TURN SCREEN BLACK
40BB: 8D 1A D0 01390           STA COLBAK
40BE: 68        01400          PLA          ; RESTORE THE REGISTERS & LEAVE
40BF: A8        01410          TAY
40C0: 68        01420          PLA
40C1: AA        01430          TAX
40C2: 68        01440          PLA
40C3: 40        01450          RTI
                01460 *
40C4: 00 00 10
40C7: 10 10 10
40CA: 38 38     01470 IMAGE    .HS 0000101010103838
40CC: 38 38 7C
40CF: 7C 7C FE
40D2: FE 54     01480          .HS 38387C7C7CFEFE54
40D4: 54 00 00 01490           .HS 540000
40D7: 00 00 00
40DA: 00 00 02
40DD: 02 02     01500 HOFF     .HS 0000000000020202
40DF: 00 FE FE
40E2: FE 00 00
40E5: 00 00     01510          .HS 00FEFEFE00000000
40E7: 00 00 00
40EA: 00 00 02
40ED: FE 00     01520 VOFF     .HS 000000000002FE00
40EF: 00 02 FE
40F2: 00 00 02
40F5: FE 00     01530          .HS 0002FE000002FE00
40F7: 44 76 B8
40FA: 3A        01540 COLOR    .HS 4476B83A
40FB: 80 41     01550 DLIST    .HS 8041
40FD: FB 40     01560          .DA DLIST
40FF:           01570 PLAYRH   .BS 1
4100:           01580 PLAYRV   .BS 1
```

Using DLIs to Create Animation

The third example shows a motorboat crossing water. A kernel controls an eight-scan line segment at the surface so that it appears that the water has waves. The interrupt occurs on the fourth text line. Essentially, each bit of an eight-bit random number is sequentially shifted right into the carry bit while in a loop. If the bit is set, the routine changes the background color register to light blue. If it isn't, it remains dark blue. Once the eight scan lines are drawn, the routine sets the background color to dark blue for the remainder of the screen. A new wave pattern is picked every eight VBlanks; otherwise, the waves would change too quickly.



DLI4.SRC

The white motorboat consists of two players set side by side. It is moved in the VBlank interrupt routine which also provides the new random number every eighth cycle. The sequence is clocked through the internal clock at $14 called RTCLOC. ANDing the value with #$07 produces a positive value if you don't have an exact even multiple of eight because some combination of the lowest three bits (1-7) is set. In that case, the routine branches to a new random number.

There are many other possible examples using kernels. Advanced programmers might attempt to modify the second example so that a non-joystick-controlled player might have its horizontal position changed during the horizontal scan for each line. It is possible, using tight timing loops, to reuse the player on the same scan line.

```
                00010 *DLI EXAMPLE - BOAT & WAVES - DAN PINAL
                00015 *SYSTEM EQUATES
0014:           00020 RTCLOC    .EQ $14
0200:           00030 VDSLST    .EQ $200
022F:           00040 SDMCTL    .EQ $22F
0230:           00050 SDLSTL    .EQ $230
02C0:           00060 PCOLR0    .EQ $2C0
02C1:           00070 PCOLR1    .EQ $2C1
02C6:           00080 COLOR2    .EQ $2C6
D000:           00090 HPOSP0    .EQ $D000
D001:           00100 HPOSP1    .EQ $D001
D018:           00110 COLPF2    .EQ $D018
D01D:           00120 GRACTL    .EQ $D01D
D20A:           00130 RANDOM    .EQ $D20A
D407:           00140 PMBASE    .EQ $D407
D40A:           00150 WSYNC     .EQ $D40A
D40E:           00160 NMIEN     .EQ $D40E
E45C:           00170 SETVBV    .EQ $E45C
E462:           00180 XITVBV    .EQ $E462
00F0:           00190 PAGE0     .EQ $F0
5000:           00200 PDATA     .EQ $5000
5400:           00210 PLAYER0   .EQ PDATA+$400
5500:           00220 PLAYER1   .EQ PDATA+$500
                00230 ;
                00240           .OR $4000
                00250 START
4000: A9 03     00260           LDA #$03    ; SET UP PLAYER/MISSILE GRAPHICS
```

```
4002: 8D 1D D0 00270          STA GRACTL
4005: A9 50    00280          LDA /PDATA
4007: 8D 07 D4 00290          STA PMBASE
400A: A9 3E    00300          LDA #$3E
400C: 8D 2F 02 00310          STA SDMCTL
400F: A9 00    00320          LDA #$00    ; CLEAR PLAYER RAM
4011: AA       00330          TAX
               00340 .1
4012: 9D 00 54 00350          STA PLAYER0,X
4015: 9D 00 55 00360          STA PLAYER1,X
4018: CA       00370          DEX
4019: D0 F7    00380          BNE .1
401B: A2 07    00390          LDX #$07
               00400 .2
401D: BD C2 40 00410          LDA LHALF,X ;PUT LEFT HALF OF BOAT ON SCREEN
4020: 9D 3C 54 00420          STA PLAYER0+$3C,X
4023: BD CA 40 00430          LDA RHALF,X ; PUT RIGHT HLAF OF BOAT ON SCREEN
4026: 9D 3C 55 00440          STA PLAYER1+$3C,X
4029: CA       00450          DEX
402A: 10 F1    00460          BPL .2
402C: A9 94    00470          LDA #$94    ; BLUE
402E: 8D C6 02 00480          STA COLOR2  ; TEXT BACKGROUND
4031: A9 0E    00490          LDA #$0E    ; WHITE
4033: 8D C0 02 00500          STA PCOLR0  ; MAKE BOAT WHITE
4036: 8D C1 02 00510          STA PCOLR1
4039: A9 00    00520          LDA #$00    ; SET HORIZONTAL POS. TO 0
403B: 8D D4 40 00530          STA POH
403E: A9 08    00540          LDA #$08    ; RIGHT HALF OF BOAT HORIZ. POS. IS 8 GREATER -
4040: 8D D5 40 00550          STA P1H     ; - THAN LEFT HALF
4043: AD 30 02 00560          LDA SDLSTL  ; MAKE A PAGE 0 POINTER TO CURRENT DISPLAY LIST
4046: 85 F0    00570          STA PAGE0
4048: AD 31 02 00580          LDA SDLSTL+1
404B: 85 F1    00590          STA PAGE0+1
404D: A0 08    00600          LDY #$08    ; FIND THE 4TH TEXT LINE
404F: B1 F0    00610          LDA (PAGE0),Y
4051: 09 80    00620          ORA #%10000000  ; TURN ON HIGH BIT FOR INTERRUPT
4053: 91 F0    00630          STA (PAGE0),Y   ; STORE IT BACK
4055: A9 00    00640          LDA #$00
4057: 8D D2 40 00650          STA WAVE
405A: A9 73    00660          LDA #DLI    ; STORE ADDRESS OF DLI IN PAGE 2 VECTOR
405C: 8D 00 02 00670          STA VDSLST
405F: A9 40    00680          LDA /DLI
4061: 8D 01 02 00690          STA VDSLST+1
4064: A9 C0    00700          LDA #%11000000   ; TELL ANTIC TO OK DLI
4066: 8D 0E D4 00710          STA NMIEN
4069: A9 07    00720          LDA #$07    ; SET UP DEFFERRED VBLANK
406B: A0 9B    00730          LDY #VBI
406D: A2 40    00740          LDX /VBI
406F: 20 5C E4 00750          JSR SETVBV
4072: 60       00760          RTS         ; THIS SHOULD RTS BACK TO YOUR ASSEMBLER DEBUGGER
               00770 DLI
4073: 48       00780          PHA
4074: 8A       00790          TXA
4075: 48       00800          PHA
4076: 98       00810          TYA
4077: 48       00820          PHA
4078: A9 A2    00830          LDA #$A2    ; DARK BLUE
407A: A0 A6    00840          LDY #$A6    ; LIGHT BLUE
407C: A2 08    00850          LDX #$08    ; WE ARE GOING TO PLAY WITH THE NEXT 8 SCAN LINES
               00860 .1
407E: 4E D3 40 00870          LSR TWAVE   ; CHECK NEXT BIT OF OUR RANDOM NUMBER
```

```
4081: 8D 0A D4 00880          STA WSYNC   ; WAIT TILL BEAM OFFSCREEN
4084: 8D 18 D0 00890          STA COLPF2  ; ASSUME DARK BLUE
4087: 90 03    00900          BCC .2      ; IF RANDOM # BIT WAS OFF
4089: 8C 18 D0 00910          STY COLPF2  ; IF ON TURN LIGHT BLUE
               00920 .2
408C: CA       00930          DEX
408D: D0 EF    00940          BNE .1      ; TILL ALL 8 LINES DONE
408F: 8D 0A D4 00950          STA WSYNC   ; WAIT TILL OFF SCREEN
4092: 8D 18 D0 00960          STA COLPF2  ; TURN SCREEN DARK BLUE FOR REMAINDER OF SCREEN
4095: 68       00970          PLA         ; RESTORE REGISTERS & LEAVE
4096: A8       00980          TAY
4097: 68       00990          PLA
4098: AA       01000          TAX
4099: 68       01010          PLA
409A: 40       01020          RTI
               01030 VBI
409B: A5 14    01040          LDA RTCLOC
409D: 29 07    01050          AND #$07    ; A NEW WAVE PATTERN IS PICKED EVERY 8TH VBLANK -
409F: D0 06    01060          BNE .1      ; - OTHERWISE IT WOULD CHANGE TOO FAST
40A1: AD 0A D2 01070          LDA RANDOM  ; GET A NEW WAVE VALUE
40A4: 8D D2 40 01080          STA WAVE
               01090 .1
40A7: AD D2 40 01100          LDA WAVE    ; GET CURRENT WAVE VALUE
40AA: 8D D3 40 01110          STA TWAVE   ; PASS TO DLI VARIABLE
40AD: EE D4 40 01120          INC POH     ; MOVE BOAT ACCROSS SCREEN
40B0: EE D5 40 01130          INC P1H
40B3: AD D4 40 01140          LDA POH
40B6: 8D 00 D0 01150          STA HPOSP0  ; TELL ANTIC NEW POS.
40B9: AD D5 40 01160          LDA P1H
40BC: 8D 01 D0 01170          STA HPOSP1
40BF: 4C 62 E4 01180          JMP XITVBV
               01190 ;
               01200 LHALF
40C2: 00 00 60
40C5: EF EF 5F
40C8: 40 C0    01210          .HS 000060EFEF5F40C0
               01220 RHALF
40CA: 80 40 20
40CD: FF FC F0
40D0: 00 00    01230          .HS 804020FFFCF00000
               01240 ;
40D2:          01250 WAVE     .BS 1
40D3:          01260 TWAVE    .BS 1
40D4:          01270 POH      .BS 1
40D5:          01280 P1H      .BS 1
```

CHAPTER 7

GAMES THAT SCROLL

An effective means of showing a much larger environment than can actually fit on the screen at any one time is to scroll the screen. Examples where only a portion of the total information can be shown are numerous, and range from the simple listing of a long BASIC program to the scanning of a large terrain map such as those used in the war game *Eastern Front.*

Perhaps its best use, however, is the dynamic feel that it gives to scrolling arcade games like *Super Cobra, Zaxxon,* and *Caverns of Mars.* These games have multi-screen worlds which scroll on or off the screen as a player's ship moves. These games show only a window or part of the entire background world at one time. They differ from games that have background stars and aliens that appear to be traveling toward you from top to bottom. Scrolling games have objects or terrain in relatively stable positions within the game's world. They can be reached by traveling to that particular section of the world.

There are two ways to scroll screen data. Most conventional micro-computers move the data through a fixed screen area. This requires an enormous memory shuffle involving many thousands of bytes. In the case of an Apple computer, rough horizontal scrolling can be achieved by shuffling all 8K of screen memory data. Other computers that are endowed with character set graphics can reduce the workload to a manageable 1K of screen data. In nearly all cases, scrolling is coarse and jerky because individual bytes of data represent either seven or eight individual pixels. Moving one byte moves many pixels at one time.

An easier method, and the one the Atari engineers chose, is to move the screen window or screen area over the data. Luckily, the ANTIC graphics microprocessor uses the Load Memory Scan (LMS) instruction to determine where it finds its screen data. A normal display list will have one LMS instruction at its beginning. The RAM area that it points to has the screen data in linear sequence. If we change the starting screen address by manipulating the operand bytes of the LMS instruction, a primitive coarse scroll can be achieved. In effect, we have moved the screen window over the data just by changing two address bytes. This is exactly equivalent to the conventional method of moving all of screen RAM.

Atari computers are also equipped with both vertical and horizontal fine scrolling registers. These enable the computer to scroll the screen in steps smaller than character or pixel size. While the effect is impressive, the technique requires implementation at a Machine language level and will be discussed later.

Coarse Vertical Scrolling

Coarse vertical scrolling, similar to the way a long BASIC program listing scrolls, is quite easy to implement from BASIC. If we consider a Graphics 0 playfield, each row of character data consists of forty characters. The second row of data begins forty bytes beyond the first row of data. When ANTIC begins fetching display data from screen RAM, it begins with the first row, using data beginning at the address in the operand of its LMS instruction. It fetches forty bytes sequentially for each of the twenty-four lines of character data.

It would be very easy to modify the two-byte operand of the LMS instruction so that it begins fetching screen data beginning with the RAM screen address of the second line or forty bytes later. To do this we need only modify the fourth and fifth bytes of our display list. The 0th, first, and second bytes are blank 8 scan-line instructions, the third is the LMS instruction, the fourth and fifth bytes are the low byte and high byte address of screen data respectively. Each time we scroll the screen upward one mode line, we need to advance the LMS address by forty bytes. You need only make sure that the low byte doesn't exceed a value of 255. If it does, you adjust the low byte by subtracting 256, and increment the high byte by one. The example below begins its scrolling at the very bottom of memory and scrolls until near the top of memory.

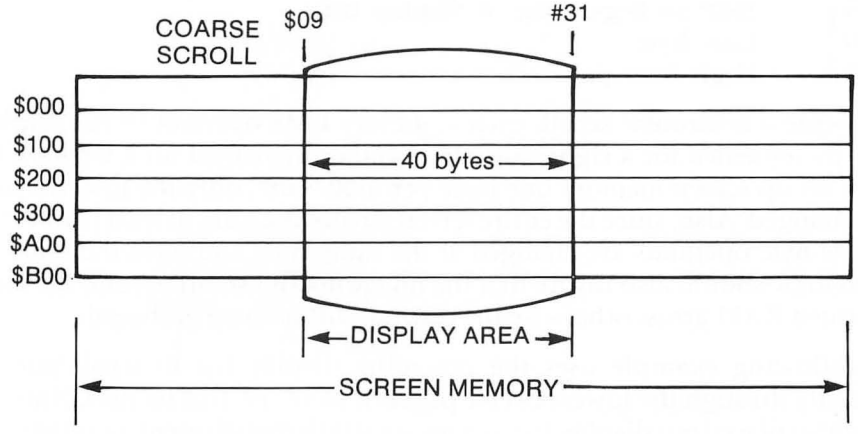
```
10 REM VERTICAL SCROLLING EXAMPLE
15 GRAPHICS 0
20 DLIST=PEEK(560)+256*PEEK(561)
30 LMSLO=DLIST+4
40 LMSHI=DLIST+5
50 SCREENLO=0:SCREENHI=0
60 REM SCROLL TO NEXT LINE
70 SCREENLO=SCREENLO+40
80 REM CHECK FOR OVERFLOW & ADJUST
90 IF SCREENLO<256 THEN 140
100 SCREENLO=SCREENLO-256
110 SCREENHI=SCREENHI+1
120 IF SCREENHI=160 THEN END
130 REM ADJUST LMS POINTERS
140 POKE LMSLO,SCREENLO
150 POKE LMSHI,SCREENHI
160 FOR DE=1 TO 50:NEXT DE
170 GOTO 70
```

As you watch the screen scroll upward, you will notice sometimes that the scrolling becomes momentarily discontinuous. This is a problem that occurs only in BASIC when scrolling is done outside the Vertical Blank period. Sometimes BASIC hasn't had time to POKE both the high and low bytes into the display list before the interrupt occurs. In the event that only one byte has been POKEd, the starting address in the LMS instruction is incorrect and the screen jerks. A more serious discontinuity occurs at 4K boundaries because ANTIC can't cross a 4K boundary using a single LMS instruction. Using LMS instructions for each mode line should solve the problem.

Coarse Horizontal Scrolling

Horizontal scrolling is much more difficult to do than vertical scrolling. The problem is that screen data is organized serially. Attempting to scroll to the right causes the computer to try to display data in the first line that would normally belong in the second line. We begin to get a snaking scroll throughout the entire display as the leftmost byte on each line will be scrolled into the rightmost position of the next higher line, and the following bytes shift one to the left.

The solution is to expand the screen data area and break it up into a series of independent horizontal data areas for each mode line. Each RAM area is still one-dimensional and serial in nature, but it extends much further than the screen shows. Since the purpose in scrolling a screen is to show more information than the screen can hold, we have allotted extra RAM for each mode line to hold this information. Now, if we move the screen window over the screen data, the data moves into view without affecting the data in any of the subsequent mode lines.



As a first step, you will need to organize your screen data by allocating a certain number of bytes of RAM for each mode line. Since the LMS operand consists of a low and a high byte, it is much easier to calculate addresses if each subsequent mode line is exactly 256 bytes long. This simplifies calculations since the high byte screen address for each mode line is a page, or one unit, apart. Since each mode line accesses a different page of screen memory, a special display list must be constructed that has an LMS for each mode line. As an example, we will horizontally scroll a BASIC mode 2 (ANTIC 7) screen. There are twelve mode lines on the screen, each using 256 bytes of memory, or a total of 3K of RAM memory. Since ANTIC displays twenty bytes per line, our world consists of nearly twenty-three screens of data arranged end to end horizontally. If we choose to use the lowest portion of RAM beginning with zero page which we know has interesting data, the display list is as follows:

```
112
112        8 blank scan lines
112
 71 )
  0 }      LMS for 0th row BASIC 2 (64+7)
  0 )      Low byte screen memory
 71 )      High byte starting zero page
  0 }      LMS for first row
  1 )      low byte
 71 )      High byte starting page one
  0 }
  2 )
  .
  .
 71 )      LMS for eleventh row
  0 }      Low byte
 11 )      High byte
 65 )      JMP to beginning of display list
  0 }      Low byte
  6 )      High byte page six
```

To execute a horizontal scroll, each and every LMS operand in the display list must be incremented for a rightward scroll and decremented for a leftward scroll. Since we set up screen memory one page per mode line, only the low byte address need be changed. Also, since the entire screen scrolls as a unit, at least in our case, all of the low byte operands are changed at the same time and have the same value. Program logic should also insure that the image doesn't scroll beyond the limits of the allocated RAM areas, otherwise the display will become garbaged.

The following example uses the preceding display list to scroll our screen horizontally through the lowest twelve pages of memory used by the OS and DOS. We have also placed our display list in page six (sixth row of our display) so that you can watch the data in the list change as the screen actually scrolls. Changes are made to each of the low byte operands of the LMS instructions via a FOR...NEXT loop. They are the fourth, seventh, tenth, etc., positions in the display list. This is at DLIST + (3*J)+1 where J goes from 1 to 12. Also, since we can't scroll into screen RAM beyond the end of each page without messing up the display, program logic dictates that the right edge of our window does not exceed 255. Therefore, the left edge must not exceed 255-20 = 235 bytes. The screen is scrolled from 0 to 235.

```
10 REM HORIZONTAL SCROLLING EXAMPLE
20 REM READ IN DISPLAY LIST TO PAGE 6
25 DLIST=1536
30 FOR I=0 TO 41
40 READ A:POKE DLIST+I,A:NEXT I
50 REM TELL ANTIC WHERE DISPLAY LIST IS
60 POKE 560,0:POKE 561,6
70 REM SCROLL SCREEN HORIZONTALLY
80 FOR I=0 TO 235
```

```
90 FOR J=1 TO 12
100 POKE DLIST+(3*J)+1,I
110 NEXT J
115 FOR DE=1 TO 75:NEXT DE
120 NEXT I
130 FOR DE=1 TO 1000:NEXT DE
140 GOTO 80
200 DATA 112,112,112,71,0,0,71,0,1,71,0,2,71,0,3,71,0,4,71,0,5,71,0,6
210 DATA 71,0,7,71,0,8,71,0,9,71,0,10,71,0,11,65,0,6
```

Obviously, the technique doesn't produce smooth wraparound, although the program repeats itself by beginning again at the left edge of screen RAM. The fault isn't with the technique but in the screen data. If you want wraparound, the data on the last screen must match that of the first screen. This would mean in our example that the last twenty bytes of each line exactly match the first twenty bytes of each line.

It wouldn't be difficult to combine horizontal scrolling with vertical scrolling to get diagonal scrolling. Since we achieve scrolling by adding or subtracting one from the LMS operand, and vertical scrolling by adding or subtracting the line length from the LMS operand, diagonal scrolling occurs when both operations are done simultaneously. If we wish to scroll down and to the left we would add 256 bytes to the high byte and 1 byte to the low byte of each scan line. While this might appear to be a simple procedure in this special case, any other configuration of screen RAM will involve two-byte additions.

Fine Scrolling

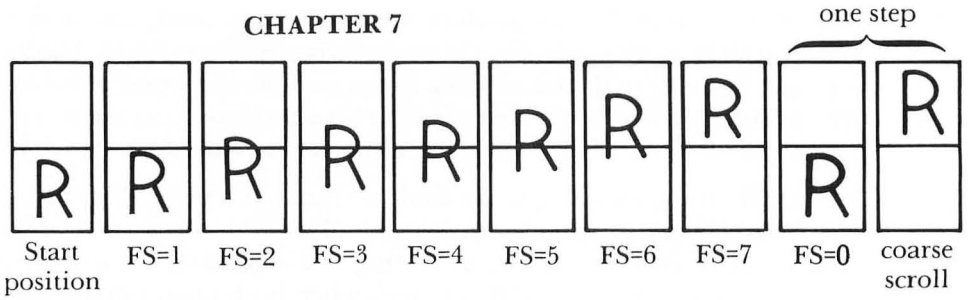
We can create much finer scrolling in steps smaller than pixel or character size by enabling the fine scrolling registers. There are two of these, one for horizontal scrolling (HSCROL) at $D404, and one for vertical scrolling (VSCROL) at $D405. They are enabled by setting appropriate bits in the display list instruction bytes for the mode lines in which we want fine scrolling. Vertical fine scrolling is enabled by setting bit 5 in the instruction bit. Similarly, horizontal fine scrolling is enabled by setting bit 4 in the instruction byte.

Function	set bit	add decimal	add hex
Load Memory Scan	6	64	40
Vertical scroll	5	32	20
Horizontal scroll	4	16	10

For example, if we were to enable fine horizontal scrolling in our example above, each of the LMS instructions would be 64+7+16 = 87.

The two fine scroll registers each have a limited range equal to 16 scan lines (0-15) in the vertical direction, and 16 color clocks (0-15) in the horizontal direction. If we

attempt to scroll beyond these values, ANTIC simply ignores the higher bits of the scroll registers. In order to achieve fine scrolling over a wider range, we need to combine fine scrolling with coarse scrolling. The technique is to fine scroll the image until the amount of fine scrolling equals the size of the pixel or character. Then you reset the fine scrolling register back to zero and coarse scroll the screen one unit. An example of fine scrolling in the vertical direction is shown in the following picture:

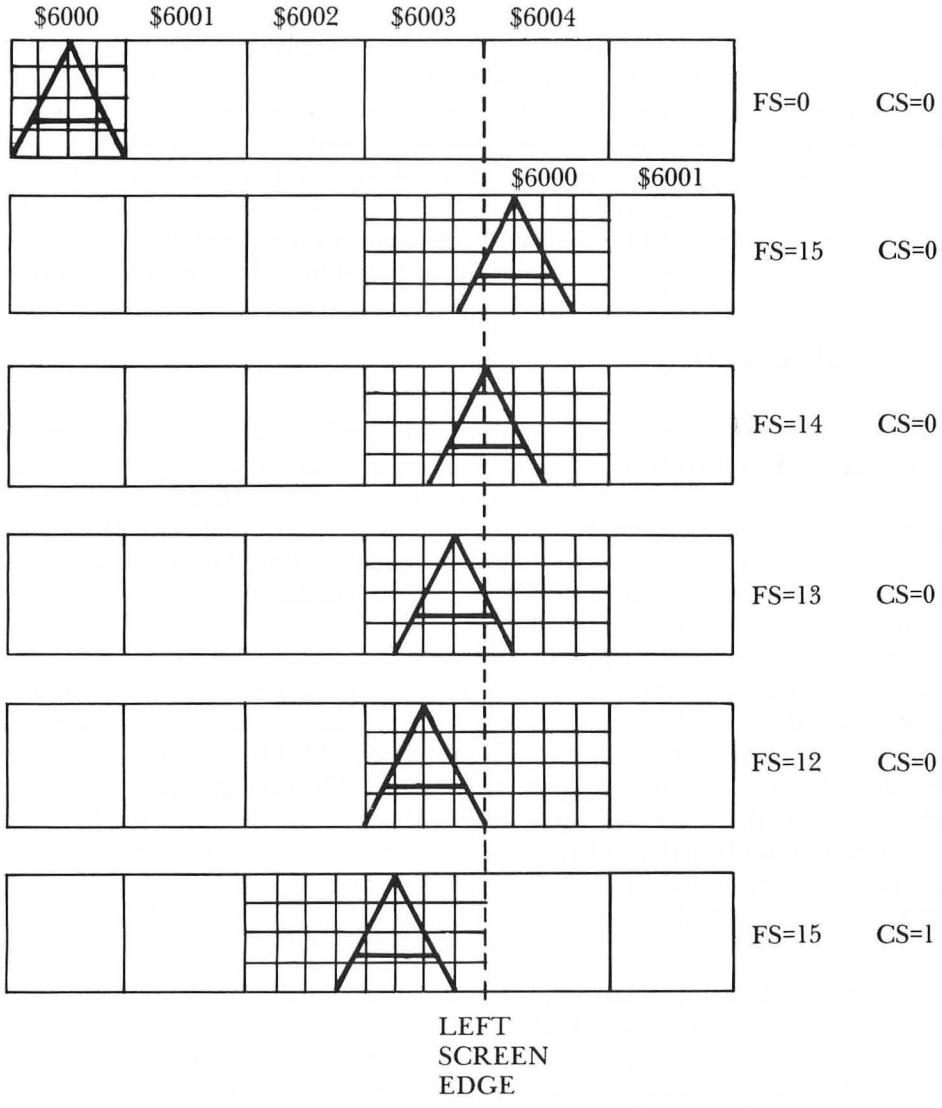


A minor problem occurs when you attempt to fine scroll data into the bottom mode line. Images tend to pop in suddenly rather than scroll in smoothly because there is no buffer of data for the next line. To get proper fine scrolling you will need to dedicate one mode line to act as a buffer. This can be done by simply not setting the vertical scroll bit in the display list instruction in the last mode line. The window will now scroll without the unpleasant jerk, but will be shortened by one mode line.

Fine scrolling in the horizontal direction is complicated by the fact that ANTIC sets aside a buffer on either end of the mode line so that portions of the pixel or character can scroll on or off the screen smoothly. When fine scrolling is enabled in the horizontal direction, ANTIC fetches more information (8 bytes/mode line) than the normal playfield (160 color clocks wide. It sets aside a buffer of sixteen color clocks on each side and retrieves forty-eight bytes of information per line rather than the usual forty bytes. This is usually not a problem if the programmer has organized his screen data in long horizontal rows. If, on the other hand, the fine scroll register is set to zero, the window is actually lookng at the sixteen color clock buffer rather than the first byte in the display for that mode line. For example, in a BASIC mode two (ANTIC 7) line where each character is eight color clocks wide, ANTIC places the first two bytes of the mode line in the buffer so that the first two bytes or sixteen color clocks aren't displayed in the screen window. BASIC mode 0 (ANTIC 2) and ANTIC mode four displays, that use characters only four color clocks wide, will be offset by four bytes. This problem can be virtually corrected by advancing the vertical fine scroll register by fifteen color clocks. You will still be off by one color clock, but since you are planning to scroll the screen anyway, it doesn't matter.

The example illustrated below shows what is involved in fine scrolling ANTIC 2 (BASIC 0) or ANTIC 4 characters horizontally. Each of these characters is only four color clocks wide so that it requires only four fine scroll increments before you need

to coarse scroll and reset the fine scroll register. The fine scroll register goes backward from 15 to 12 before being reset to 15. The top row shows that ANTIC actually places the first character into the buffer rather than into the first visible screen position if the fine scroll register is set to zero. The second row shows the correction obtained by setting the fine scroll register to 15. While it seems that a value of 16 should correct it totally, in fact the high bit would be ignored, and you would obtain the result of the top row.



Whether the movement be vertical, horizontal, or diagonal, the most difficult aspect of using Machine language to scroll the screen is calculating the LMS addresses for each of the mode lines. Multiplication of numbers other than powers of two, requires a complicated and time-consuming subroutine. Fortunately, several special or contrived scrolling cases make the calculation fairly easy. We will discuss these situations first.

Vertical Scrolling

Pure vertical scrolling is obviously the easiest case. It requires only one LMS instruction because screen memory is continous throughout the display. Still, if we had to calculate the starting address of the display from scratch each time we scrolled, we would still need an elaborate multiplication subroutine because multiplication by forty bytes or #$28 is not an easy feat. Luckily, rough scrolling is usually done a line at a time. Since we know the current address of the screen, we need only add forty bytes to this address to scroll the screen upwards, or subtract forty bytes to scroll the screen downwards. This only requires a double-byte add or double-byte subtraction.

Horizontal Scrolling

Pure horizontal scrolling, on the other hand, can become very complicated and require dozens of multiplications, if the screen size isn't a special case. The most common special case is one where each mode line is exactly 256 bytes or one page in memory. The LMS address for the subsequent mode lines are exactly $100 apart in memory. This becomes very convenient since you don't need to do an addition or subtraction on each of the current LMS addresses in order the rough scroll. Instead you can merely store the new value of the horizontal rough scroll offset into each of the low byte addresses for the LMS instructions. If you are in GRAPHICS 1 (ANTIC 6), this means that you have to complete twenty-four store operations in a simple loop. Perhaps the best example of this is our scrolling game example in the final section of this chapter. There are twenty-two rows of ANTIC 6 characters, each one page in length for a total of 5 ½K of screen memory. The variable XS determines the rough scroll position and the variable FS, the fine scroll position. Fine scroll naturally progresses from 0 to 7 before there is a need to increment XS. However, as we discussed earlier, the actual value placed into the fine scroll hardware register is HSCROL = 15–FS. Wraparound in this example is at XS = 235. When XS exceeds 235, XS is reset to 0.

All twenty-two low byte addresses are updated in the display list during VBlank. The first of these addresses is at NDLIST+8. The next address is three bytes later. We can take advantage of indirect addressing using the Y register if we increase the Y register by 3 between store operations. The code below illustrates the technique.

```
        LDY #$00          ;COUNTER
.4      LDA XS            ;POSITION AT SCREEN LEFT
        STA NDLIST+8,Y    ;LOACATION OF FIRST LOW BYTE ADDRESS
        INY               ;LOW BYTES ARE THREE APART
        INY
        INY
        CPY #$4B          ;END OF LIST?
        BNE .4            ;NEXT ELEMENT
```

Obviously, if we choose to include vertical scrolling as well, we need only increment each of the high byte addresses of the LMS instruction to rough scroll the screen upwards by one mode line. The main problem is that a screen 256 bytes wide (six plus screens) uses a very great amount of screen memory. A depth of two screens in GRAPHICS 1 (ANTIC 6) would require 12K, and six screens would require 36K. This is an enourmous amount of screen memory in addition to the game code, even for a 48K Atari.

Eight Way Scrolling — Special Case

A simple but contrived eight-way scrolling example could be developed that has a width of 128 bytes. The depth of course will be determined by the amount of screen memory available. The advantage of a 128-byte width over a 256-byte width is that it will allow a deeper scrolling playfield without requiring a complicated mutiplication subroutine. Even so, the calculation is not as simple as the example above. First, it requires calculating the display address of the initial LMS instruction based on the rough scroll position YS. Each subsequent LMS address is determined by adding $80 to the previous one, then adding the horizontal rough scroll position XS to that.

ADDRESS = SCREEN + (YS*$80) + XS

We used a trick to avoid the initial multiplication usually needed to find the first LMS address. Since the high byte of the address increases every time our vertical rough scroll position (YS) increases by two, then:

SCHI = SCREEN/256 + YS/2

Odd values of YS set the carry bit after the division. If that occurs #$80 is added to the low byte of the first instruction. Afterwards the horizontal rough scroll offset XS is added.

SCLO = Screen address low byte + #$80 + XS (if carry set after division)

SCLO = Screen address low byte + XS (if carry clear after division)

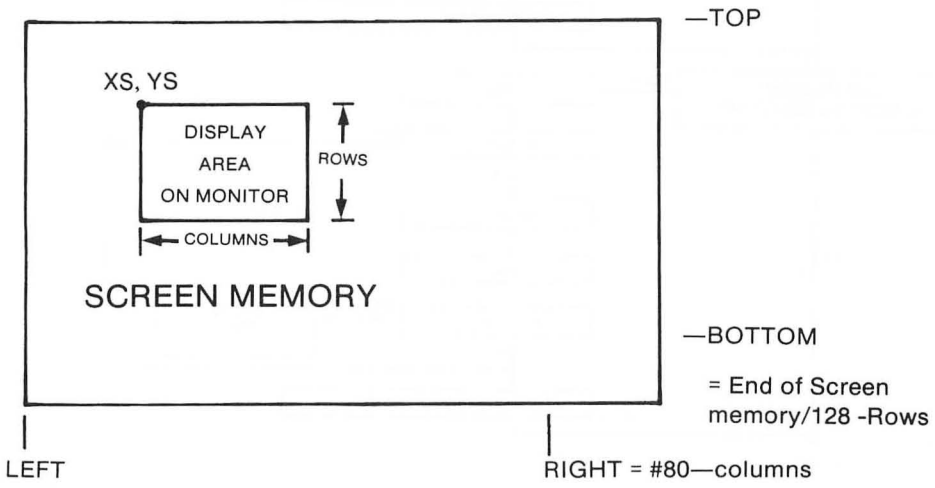
Since each of the mode lines are scrolled equally horizontally, you only need to add #$80 in a double-byte addition to find the starting address of the next mode line. Again, this whole operation can be performed using indirect indexing. The first

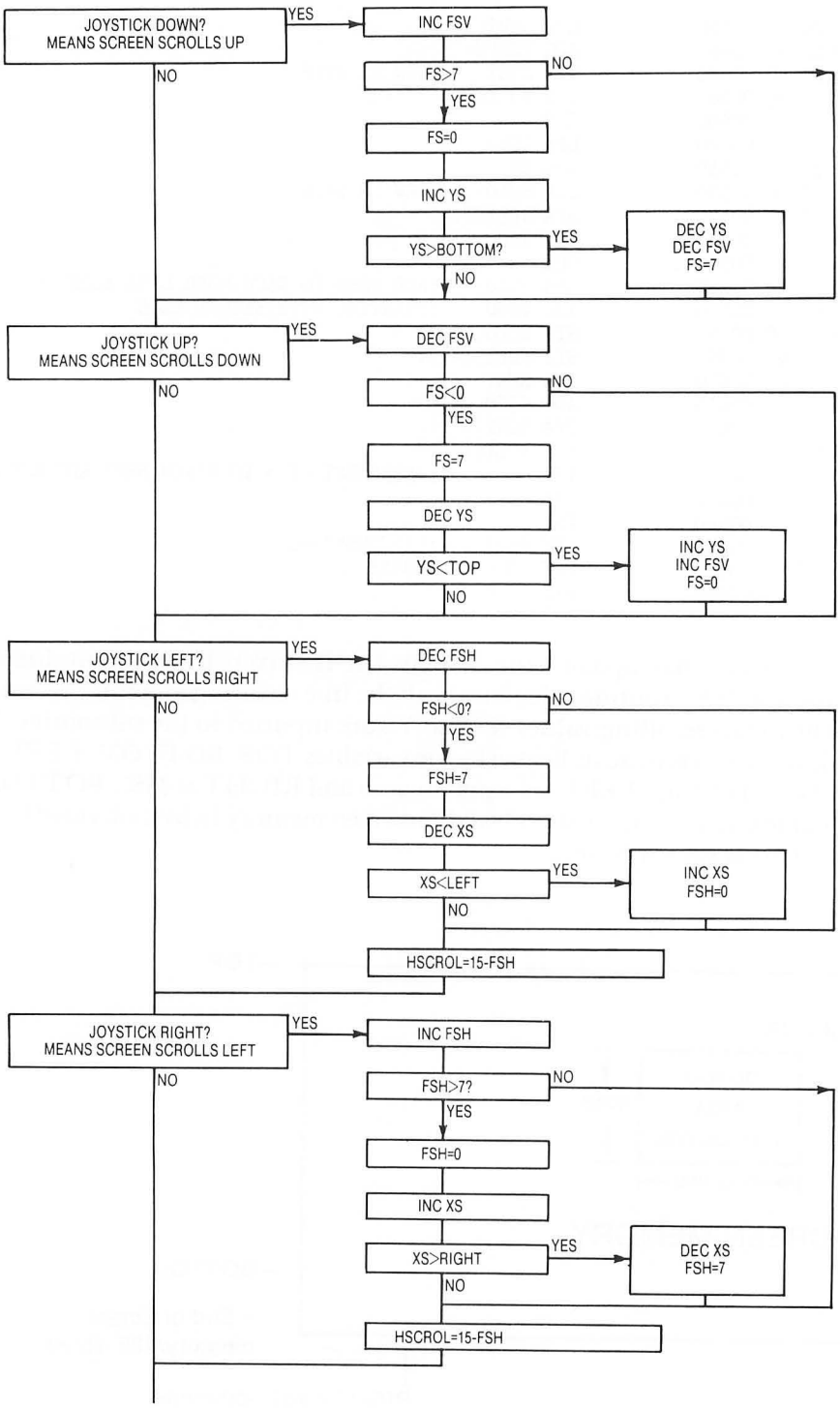
LMS low byte address is at NDLIST+7, and the high byte address is at NDLIST+8. We can index both of these addresses using the Y register, then increment the Y register by 3 in a loop to reach subsequent LMS address pairs. The routine exits the loop when all of the mode line addresses have been calculated or when the Y register equals (#ROWS * 3)-3.

```
                    00010 *TEST CASE FOR MODIFYING DLIST 8 WAY SCROLLING
                    00020          .OR $4000
5000:               00030 SCREEN   .EQ $5000
4900:               00040 NDLIST   .EQ $4900
4000: 70 70 70
4003: 47 00 50
4006: 47 80 50 00050 DLIST    .HS 707070470050478050
4009: 47 00 51
400C: 47 80 51
400F: 47 00 52 00060          .HS 470051478051470052
4012: 47 80 52
4015: 47 00 53
4018: 47 80 53 00070          .HS 478052470053478053
401B: 47 00 54
401E: 47 80 54
4021: 47 00 48 00080          .HS 470054478054470048
4024: 47 80 48
4027: 41 00 49 00090          .HS 478048410049
                    00100 *VARIABLES
402A:               00110 XS       .BS 1
402B:               00120 YS       .BS 1
402C:               00130 SCLO     .BS 1
402D:               00140 SCHI     .BS 1
402E:               00150 TEMPH    .BS 1
                    00160 *MAIN PROGRAM
402F: A9 05         00170 START    LDA #$05    ;TEST VALUES - INPUT YOUR OWN XS,YS
4031: 8D 2A 40 00180          STA XS
4034: A9 03         00190          LDA #$03
4036: 8D 2B 40 00200          STA YS
                    00210 *MOVE DLIST
4039: A2 00         00220          LDX #$00
403B: BD 00 40 00230 DLOOP    LDA DLIST,X
403E: 9D 00 49 00240          STA NDLIST,X
4041: E8            00250          INX
4042: E0 2A         00260          CPX #$2A    ;44 ELEMENTS
4044: D0 F5         00270          BNE DLOOP
4046: 20 4C 40 00280          JSR MODLIST ;MODIFY DISPLAY LIST
4049: 4C 49 40 00290 FOREVER  JMP FOREVER ;ENDLESS LOOP
                    00300 *
                    00310 *SUBROUTINE TO MODIFY DISPLAY LIST FOR 8 WAY SCROLLING
                    00320 * - 128 (#$80) BYTES WIDE
                    00330 *INPUT ROUGH SCROLL COORDINATES XS,YS
404C: A9 00         00340 MODLIST  LDA #SCREEN
404E: 8D 2C 40 00350          STA SCLO
4051: A9 50         00360          LDA /SCREEN
4053: 8D 2D 40 00370          STA SCHI
4056: AD 2B 40 00380          LDA YS
4059: 4A            00390          LSR         ;DIVIDE/2
405A: 8D 2E 40 00400          STA TEMPH
405D: 90 05         00410          BCC .1      ;SKIP IF EVEN
405F: A9 80         00420          LDA #$80    ;ODD THEN LO BYTE ADDRESS BEGINS WITH #$80
4061: 8D 2C 40 00430          STA SCLO
4064: 18            00440 .1       CLC
```

```
4065: AD 2D 40 00450          LDA SCHI
4068: 6D 2E 40 00460          ADC TEMPH
406B: 8D 2D 40 00470          STA SCHI     ;NEW HI BYTE
406E: 8D 05 49 00480          STA NDLIST+5
4071: 18       00490          CLC
4072: AD 2C 40 00500          LDA SCLO
4075: 6D 2A 40 00510          ADC XS
4078: 8D 2C 40 00520          STA SCLO     ;NEW LO BYTE
407B: 8D 04 49 00530          STA NDLIST+4
407E: A0 00    00540          LDY #$00
4080: 18       00550 .2       CLC
4081: AD 2C 40 00560          LDA SCLO     ;ADD #$80 TO EACH MODE LINE ADDRESS
4084: 69 80    00570          ADC #$80     ;STARTING WITH SECOND LINE
4086: 8D 2C 40 00580          STA SCLO
4089: 99 07 49 00590          STA NDLIST+7,Y
408C: AD 2D 40 00600          LDA SCHI
408F: 69 00    00610          ADC #$00
4091: 8D 2D 40 00620          STA SCHI
4094: 99 08 49 00630          STA NDLIST+8,Y
4097: C8       00640          INY          ;INCREMENT BY 3 TO REACH NEXT ADDRESS PAIR
4098: C8       00650          INY
4099: C8       00660          INY
409A: C0 21    00670          CPY #$21     ;(12ROWS*3)-3
409C: 90 E2    00680          BLT .2       ;DONE?
409E: 60       00690          RTS
```

The subroutine that updates the display list is driven by an eight-direction joystick routine. This routine calculates both the fine scrolling and rough scrolling values. The rough scrolling values XS and YS are inputted to the subroutine. The boundaries of the screen are indicated by the variables TOP, BOTTOM, LEFT, and RIGHT. Both TOP and LEFT are equal to xero and RIGHT is #$80. BOTTOM is user definable but must be equal or less than screen memory in bytes divided by 128. The flow chart is shown below.




JOYSTICK DOWN?
MEANS SCREEN SCROLLS UP
YES
INC FSV
FS>7
NO
YES
FS=0
INC YS
YS>BOTTOM?
YES
DEC YS
DEC FSV
FS=7
NO
NO
JOYSTICK UP?
MEANS SCREEN SCROLLS DOWN
YES
DEC FSV
FS<0
NO
YES
FS=7
DEC YS
YS<TOP
YES
INC YS
INC FSV
FS=0
NO
NO
JOYSTICK LEFT?
MEANS SCREEN SCROLLS RIGHT
YES
DEC FSH
FSH<0?
NO
YES
FSH=7
DEC XS
XS<LEFT
YES
INC XS
FSH=0
NO
NO
HSCROL=15-FSH
JOYSTICK RIGHT?
MEANS SCREEN SCROLLS LEFT
YES
INC FSH
FSH>7?
NO
YES
FSH=0
INC XS
XS>RIGHT
YES
DEC XS
FSH=7
NO
NO
HSCROL=15-FSH

Use of Lookup Tables to Determine LMS Screen Addresses

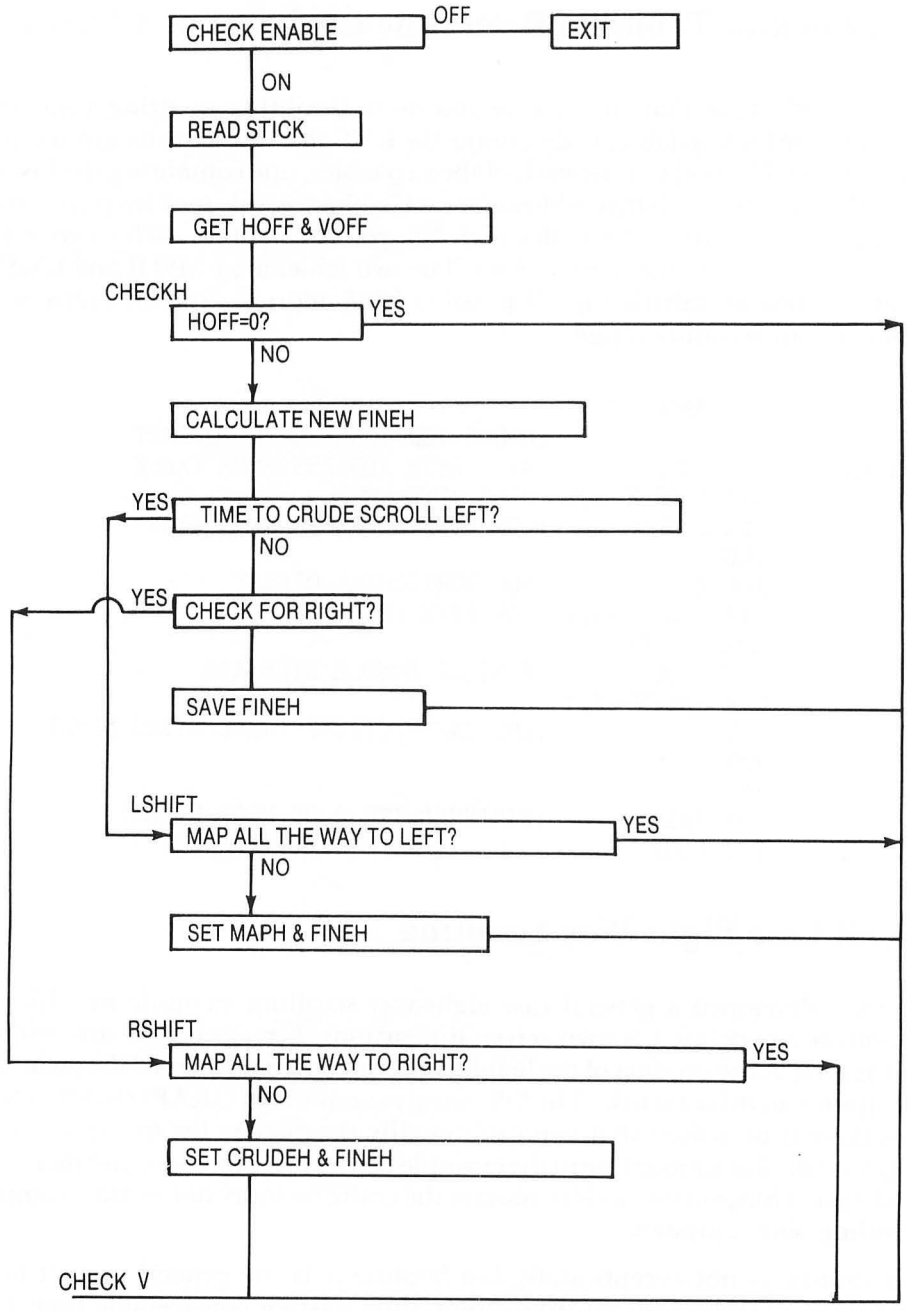
Another technique that would give you more flexibility in sizing your screen would be to use lookup tables to determine the LMS addresses for any given vertical scrolling offset YS. You can use each of the two tables, one containing the low byte address, the other the high byte address for each of the possible scrolled positions. By indexing into the start of the tables with YS, you can lookup each of your LMS addresses, then add the horizontal offset. The two tables are LMSHI and LMSLO. Both are formed by calculating all possible LMS addresses for the entire screen memory of your scrolling range.

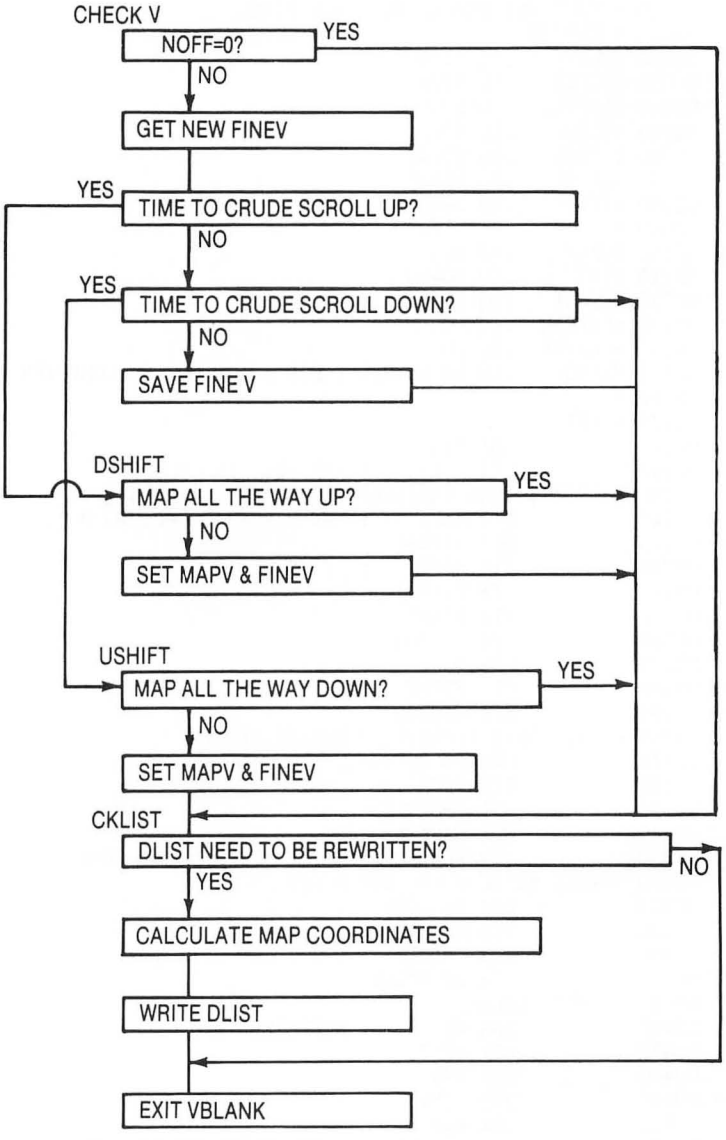
```
        LDY #$00
        LDX YS          ;ROUGH VERTICAL SCROLL OFFSET
LOOP    LDA LMSHI       ;HIGH BYTE ADDRESS FROM TABLE
        STA NDLIST+5,Y  ;HIGH BYTE LMS
        LDA LMSLO       ;LOW BYTE ADDRESS FROM TABLE
        CLC
        ADC XS          ;ADD HORIZONTAL OFFSET
        STA NDLIST+4,Y  ;LOW BYTE LMS
        LDA NDLIST+5,Y
        ADC #$00        ;REST OF DOUBLE BYTE ADD
        STA NDLIST+5,Y
        INY             ;LMS INSTRUCTIONS-THREE BYTES APART
        INY
        INY
        CPX #$16        ;FINISHEDWITH # OF MODE LINES?
        BLT LOOP
```

General Case Eight-Way Scrolling

We have developed a general case eight-way scrolling example in which the programmer can define his own screen dimensions. Screens can be any width or height as long as the product of the height and width does not exceed the memory in the computer, in this case 64K. The following example is in GRAPHICS 0 (ANTIC 2), but there is no reason that you can't modify the display list to use a different graphics mode. Each mode line in the example is 1024 bytes in width and there are 64 rows of data. Thus, you are able to observe the entire memory of the Atari computer by scrolling with a joystick.

The routine is not exceptionally fast because it is the general case. It makes extensive use of sixteen-bit multiplication, thus wasting considerable time doing calculations. We don't recommend it for standard arcade games in which the screen is updated sixty times a second, but it is quite useful in tactical war games or other types of displays in which the screen changes less frequently. Both the flowchart and code follow.


CHECK ENABLE
OFF
EXIT
ON
READ STICK
GET HOFF & VOFF
CHECKH
HOFF=0?
YES
NO
CALCULATE NEW FINEH
YES
TIME TO CRUDE SCROLL LEFT?
NO
YES
CHECK FOR RIGHT?
SAVE FINEH
LSHIFT
MAP ALL THE WAY TO LEFT?
YES
NO
SET MAPH & FINEH
RSHIFT
MAP ALL THE WAY TO RIGHT?
YES
NO
SET CRUDEH & FINEH
CHECK V


CHECK V
NOFF=0?
YES
NO
GET NEW FINEV
YES
TIME TO CRUDE SCROLL UP?
NO
YES
TIME TO CRUDE SCROLL DOWN?
NO
SAVE FINE V
DSHIFT
MAP ALL THE WAY UP?
YES
NO
SET MAPV & FINEV
USHIFT
MAP ALL THE WAY DOWN?
YES
NO
SET MAPV & FINEV
CKLIST
DLIST NEED TO BE REWRITTEN?
NO
YES
CALCULATE MAP COORDINATES
WRITE DLIST
EXIT VBLANK

```
                00010 *EIGHT WAY SCROLLING - DAN PINAL
                00015 *EQUATES
0230:           00020 SDLSTL   .EQ $230
0278:           00030 STICKO   .EQ $278
D402:           00040 DLISTL   .EQ $D402
D404:           00050 HSCROL   .EQ $D404
D405:           00060 VSCROL   .EQ $D405
E45C:           00070 SETVBV   .EQ $E45C
E462:           00080 XITVBV   .EQ $E462
                00090 *
0000:           00100 SCREEN   .EQ $0000
0400:           00110 WIDTH    .EQ $0400
0040:           00120 HEIGHT   .EQ $0040
0028:           00130 SCREENH  .EQ 40
0014:           00140 SCREENV  .EQ 20
9F60:           00150 WINDOW   .EQ $9C40+800 ; FOR A 40K OR 48K COMPUTER WITH CARTRIDGE
                00160 *
                00170 START
4000: A9 00     00180          LDA #$00
4002: 8D 5D 42  00190          STA ENABLE  ; SET VBI FLAG TO OFF
                00200 * INIT SCROLL VARIABLES TO STARTUP VALUES
4005: 8D 4E 42  00210          STA MAPH    ; SET SCREEN COORD. TO 0
4008: 8D 4F 42  00220          STA MAPH+1
400B: 8D 50 42  00230          STA MAPV    ;
400E: 8D 51 42  00240          STA MAPV+1
4011: 8D 59 42  00250          STA FINEV
4014: 8D 05 D4  00260          STA VSCROL
4017: A9 0C     00270          LDA #$0C
4019: 8D 58 42  00280          STA FINEH
401C: 8D 04 D4  00290          STA HSCROL
                00300 * INIT BASE ADDRESS TO SCREEN ADDRESS
401F: A9 00     00310          LDA #SCREEN
4021: 8D 56 42  00320          STA BASE
4024: A9 00     00330          LDA /SCREEN
4026: 8D 57 42  00340          STA BASE+1
4029: 20 BB 41  00350          JSR WRITEDL ; WRITE NEW DISPLAY LIST
                00360 * TELL ANTIC WHERE NEW DLIST IS
402C: A9 02     00370          LDA #NDLIST
402E: 8D 30 02  00380          STA SDLSTL
4031: A9 42     00390          LDA /NDLIST
4033: 8D 31 02  00400          STA SDLSTL+1
                00410 * SETUP VBLANK
4036: A9 07     00420          LDA #$07    ; DEFFERED VBI
4038: A0 45     00430          LDY #VBI
403A: A2 40     00440          LDX /VBI
403C: 20 5C E4  00450          JSR SETVBV
403F: A9 01     00460          LDA #$01
4041: 8D 5D 42  00470          STA ENABLE  ; TURN VBLANK FLAG TO ON
4044: 60        00480          RTS         ; BACK TO YOUR MONITOR
                00490 *
                00500 VBI
4045: D8        00510          CLD         ; JUST A PRECAUTION
4046: AD 5D 42  00520          LDA ENABLE  ; CHECK THE SOFTWARE FLAG
4049: D0 03     00530          BNE CKSTK   ; OK
404B: 4C B8 41  00540          JMP XVBI    ; LEAVE VBI
                00550 CKSTK
                00560 ; COPY OLD MAP VARIABLES IN CASE NEW VALUES ARE INVALID
404E: AD 58 42  00570          LDA FINEH
4051: 8D 5A 42  00580          STA NEWFH
4054: AD 59 42  00590          LDA FINEV
4057: 8D 5B 42  00600          STA NEWFV
405A: A9 00     00610          LDA #$00
```

```
405C: 8D 5C 42 00620          STA CHANGE  ; SET CHANGE FLAG TO 0
              00630 * USE JOYSTICK VALUE TO INDEX INTO TABLE OF +1,0, OR -1
405F: AE 78 02 00640          LDX STICK0
4062: BD E2 41 00650          LDA HOFFS,X
4065: 8D 52 42 00660          STA HOFF
4068: BD F2 41 00670          LDA VOFFS,X
406B: 8D 54 42 00680          STA VOFF
              00690 CHECKH
406E: AD 52 42 00700          LDA HOFF
4071: F0 6C    00710          BEQ CHECKV  ; IF 0 NO CHANGE LEAVE
              00720 * ADD OFFSET TO FINEH
4073: 18       00730          CLC
4074: AD 5A 42 00740          LDA NEWFH
4077: 6D 52 42 00750          ADC HOFF
407A: 8D 5A 42 00760          STA NEWFH
407D: C9 10    00770          CMP #$10    ; TIME TO CRUDE SCROLL?
407F: F0 0D    00780          BEQ RSHIFT
4081: C9 0B    00790          CMP #$0B    ; TIME TO CRUDE SCROLL?
4083: F0 30    00800          BEQ LSHIFT
4085: 8D 58 42 00810          STA FINEH
4088: 8D 04 D4 00820          STA HSCROL
408B: 4C DF 40 00830          JMP CHECKV  ; GO CHECK VERTICAL
              00840 RSHIFT
408E: AD 4E 42 00850          LDA MAPH    ; CHECK IF HORIZONTAL ALREADY AT 0?
4091: 0D 4F 42 00860          ORA MAPH+1
4094: F0 49    00870          BEQ CHECKV      ; CAN'T GO LESS THAN 0
4096: 38       00880          SEC
4097: AD 4E 42 00890          LDA MAPH
409A: E9 01    00900          SBC #$01
409C: 8D 4E 42 00910          STA MAPH
409F: AD 4F 42 00920          LDA MAPH+1
40A2: E9 00    00930          SBC #$00
40A4: 8D 4F 42 00940          STA MAPH+1
40A7: A9 0C    00950          LDA #$0C
40A9: 8D 58 42 00960          STA FINEH
40AC: 8D 04 D4 00970          STA HSCROL
40AF: EE 5C 42 00980          INC CHANGE  ; SET FLAG TO WRITE NEW DISPLAY LIST
40B2: 4C DF 40 00990          JMP CHECKV
              01000 LSHIFT
              01010 ; FIRST CHECK IF ALREADY AT LIMIT
40B5: AD 4E 42 01020          LDA MAPH
40B8: C9 D8    01030          CMP #WIDTH-SCREENH
40BA: D0 07    01040          BNE .1
40BC: AD 4F 42 01050          LDA MAPH+1
40BF: C9 03    01060          CMP /WIDTH-SCREENH
40C1: F0 1C    01070          BEQ CHECKV
              01080 .1
              01090 * CHANGE MAPH & SET FINE SCROLL
40C3: 18       01100          CLC
40C4: AD 4E 42 01110          LDA MAPH
40C7: 69 01    01120          ADC #$01
40C9: 8D 4E 42 01130          STA MAPH
40CC: AD 4F 42 01140          LDA MAPH+1
40CF: 69 00    01150          ADC #$00
40D1: 8D 4F 42 01160          STA MAPH+1
40D4: A9 0F    01170          LDA #$0F
40D6: 8D 58 42 01180          STA FINEH
40D9: 8D 04 D4 01190          STA HSCROL
40DC: EE 5C 42 01200          INC CHANGE  ; SET FLAG FOR CKLIST
              01210 CHECKV
              01220 * VERTICAL WORKS SAME AS HORIZONTAL
40DF: AD 54 42 01230          LDA VOFF
```

```
40E2: F0 6F    01240          BEQ CKLIST  ; LEAVE IF NO CHANGES TO MAKE
               01250 * ADD OFFSET TO FINE SCROLL VALUE
40E4: 18       01260          CLC
40E5: 6D 5B 42 01270          ADC NEWFV
40E8: 8D 5B 42 01280          STA NEWFV
               01290 * CHECK TO SEE IF IT'S TIME TO CRUDE SCROLL
40EB: C9 08    01300          CMP #$08
40ED: F0 0D    01310          BEQ DSHIFT
40EF: C9 FF    01320          CMP #$FF
40F1: F0 39    01330          BEQ USHIFT
40F3: 8D 59 42 01340          STA FINEV
40F6: 8D 05 D4 01350          STA VSCROL
40F9: 4C 53 41 01360          JMP CKLIST
               01370 DSHIFT
               01380 * CHECK TO SEE IF MAP IS AT LIMIT
40FC: AD 50 42 01390          LDA MAPV
40FF: C9 2C    01400          CMP #HEIGHT-SCREENV
4101: D0 07    01410          BNE .1
4103: AD 51 42 01420          LDA MAPV+1
4106: C9 00    01430          CMP /HEIGHT-SCREENV
4108: F0 49    01440          BEQ CKLIST
               01450 .1
               01460 * SET MAP VERT. OFFSET & RESET FINE SCROLL VALUE
410A: 18       01470          CLC
410B: AD 50 42 01480          LDA MAPV
410E: 69 01    01490          ADC #$01
4110: 8D 50 42 01500          STA MAPV
4113: AD 51 42 01510          LDA MAPV+1
4116: 69 00    01520          ADC #$00
4118: 8D 51 42 01530          STA MAPV+1
411B: AD 5B 42 01540          LDA NEWFV
411E: 29 07    01550          AND #$07
4120: 8D 59 42 01560          STA FINEV
4123: 8D 05 D4 01570          STA VSCROL
4126: EE 5C 42 01580          INC CHANGE
4129: 4C 53 41 01590          JMP CKLIST
               01600 USHIFT
               01610 * CHECK FOR MAP AT LIMIT
412C: AD 50 42 01620          LDA MAPV
412F: 0D 51 42 01630          ORA MAPV+1
4132: F0 1F    01640          BEQ CKLIST
4134: 38       01650          SEC
4135: AD 50 42 01660          LDA MAPV
4138: E9 01    01670          SBC #$01
413A: 8D 50 42 01680          STA MAPV
413D: AD 51 42 01690          LDA MAPV+1
4140: E9 00    01700          SBC #$00
4142: 8D 51 42 01710          STA MAPV+1
4145: AD 5B 42 01720          LDA NEWFV
4148: 29 07    01730          AND #$07
414A: 8D 59 42 01740          STA FINEV
414D: 8D 05 D4 01750          STA VSCROL
4150: EE 5C 42 01760          INC CHANGE
               01770 CKLIST
4153: AD 5C 42 01780          LDA CHANGE; CHECK IF MAP HORIZONTAL OR VERTICAL HAS BEEN CHANGE
4156: F0 60    01790          BEQ XVBI   ; NO NEED TO REWRITE DISPLAY LIST IF NO CHANGE
               01800 * SET UP TO MULTIPLY WIDTH X MAP VERTICAL OFFSET
4158: A9 00    01810          LDA #WIDTH
415A: 8D 4A 42 01820          STA RESULT
415D: A9 04    01830          LDA /WIDTH
415F: 8D 4B 42 01840          STA RESULT+1
4162: A9 00    01850          LDA #$00    ; OVERFLOW SHOULD NOT BE NEEDED BUT ITS HERE ANYWAY
```

```
4164: 8D 4C 42 01860          STA OVERFL
4167: 8D 4D 42 01870          STA OVERFL+1
416A: A2 11    01880          LDX #$11
416C: 18       01890          CLC
               01900 * THIS IS A 16 BIT MULTIPLY TO CALCULATE OFFSET FROM STARTING MAP
               01910 MULT16
416D: 6E 4D 42 01920          ROR OVERFL+1
4170: 6E 4C 42 01930          ROR OVERFL
4173: 6E 4B 42 01940          ROR RESULT+1
4176: 6E 4A 42 01950          ROR RESULT
4179: 90 13    01960          BCC .1
417B: 18       01970          CLC
417C: AD 50 42 01980          LDA MAPV
417F: 6D 4C 42 01990          ADC OVERFL
4182: 8D 4C 42 02000          STA OVERFL
4185: AD 51 42 02010          LDA MAPV+1
4188: 6D 4D 42 02020          ADC OVERFL+1
418B: 8D 4D 42 02030          STA OVERFL+1
               02040 .1
418E: CA       02050          DEX
418F: D0 DC    02060          BNE MULT16
               02070 * NOW SCREEN ADDRESS IS ADDED
4191: 18       02080          CLC
4192: A9 00    02090          LDA #SCREEN
4194: 6D 4A 42 02100          ADC RESULT
4197: 8D 56 42 02110          STA BASE
419A: A9 00    02120          LDA /SCREEN
419C: 6D 4B 42 02130          ADC RESULT+1
419F: 8D 57 42 02140          STA BASE+1
               02150 * NOW THE COLUMN OFFSET IS ADDED TO FORM THE NEW BASE ADDRESS
41A2: 18       02160          CLC                                    FOR THE TOP
41A3: AD 56 42 02170          LDA BASE
41A6: 6D 4E 42 02180          ADC MAPH
41A9: 8D 56 42 02190          STA BASE
41AC: AD 57 42 02200          LDA BASE+1
41AF: 6D 4F 42 02210          ADC MAPH+1
41B2: 8D 57 42 02220          STA BASE+1
               02230 *
41B5: 20 BB 41 02240          JSR WRITEDL ; WRITE NEW DISPLAY LIST
               02250 XVBI
41B8: 4C 62 E4 02260          JMP XITVBV
               02270 *
               02280 WRITEDL
               02290 * SPOT POINTS TO THE FIRST ADDRESS IN THE DISPLAY LIST
41BB: A2 00    02300          LDX #$00
               02310 .1
               02320 * STORE NEW ADDRESS IN DISPLAY LIST
41BD: AD 56 42 02330          LDA BASE
41C0: 9D 06 42 02340          STA SPOT,X
41C3: AD 57 42 02350          LDA BASE+1
41C6: 9D 07 42 02360          STA SPOT+1,X
               02370 * ADD MAP WIDTH TO GET ADDRESS OF NEXT LINE
41C9: 18       02380          CLC
41CA: A9 00    02390          LDA #WIDTH
41CC: 6D 56 42 02400          ADC BASE
41CF: 8D 56 42 02410          STA BASE
41D2: A9 04    02420          LDA /WIDTH
41D4: 6D 57 42 02430          ADC BASE+1
41D7: 8D 57 42 02440          STA BASE+1
               02450 * SET TO POINT TO NEXT SET OFF ADDRESSES IN DLIST
41DA: E8       02460          INX
41DB: E8       02470          INX
```

```
41DC: E8          02480           INX
41DD: E0 3C       02490           CPX #60
41DF: D0 DC       02500           BNE .1
41E1: 60          02510           RTS
                  02520 *
                  02530 HOFFS
41E2: 00 00 00
41E5: 00 00 01
41E8: 01 01       02540           .HS 0000000000010101
41EA: 00 FF FF
41ED: FF 00 00
41F0: 00 00       02550           .HS 00FFFFFF00000000
                  02560 VOFFS
41F2: 00 00 00
41F5: 00 00 01
41F8: FF 00       02570           .HS 000000000001FF00
41FA: 00 01 FF
41FD: 00 00 01
4200: FF 00       02580           .HS 0001FF000001FF00
                  02590 NDLIST
4202: 70 70 70 02600              .HS 707070
4205: 72          02610           .HS 72
                  02620 SPOT
4206: 00 00       02630           .HS 0000
4208: 72 00 00 02640              .HS 720000
420B: 72 00 00 02650              .HS 720000
420E: 72 00 00 02660              .HS 720000
4211: 72 00 00 02670              .HS 720000
4214: 72 00 00 02680              .HS 720000
4217: 72 00 00 02690              .HS 720000
421A: 72 00 00 02700              .HS 720000
421D: 72 00 00 02710              .HS 720000
4220: 72 00 00 02720              .HS 720000
4223: 72 00 00 02730              .HS 720000
4226: 72 00 00 02740              .HS 720000
4229: 72 00 00 02750              .HS 720000
422C: 72 00 00 02760              .HS 720000
422F: 72 00 00 02770              .HS 720000
4232: 72 00 00 02780              .HS 720000
4235: 72 00 00 02790              .HS 720000
4238: 72 00 00 02800              .HS 720000
423B: 72 00 00 02810              .HS 720000
423E: 52 00 00 02820              .HS 520000
4241: 42          02830           .HS 42
4242: 60 9F       02840           .DA WINDOW
4244: 02 02 02 02850              .HS 020202
4247: 41          02860           .HS 41
4248: 02 42       02870           .DA NDLIST
                  02880 *
424A:             02890 RESULT    .BS 2
424C:             02900 OVERFL    .BS 2
424E:             02910 MAPH      .BS 2
4250:             02920 MAPV      .BS 2
4252:             02930 HOFF      .BS 2
4254:             02940 VOFF      .BS 2
4256:             02950 BASE      .BS 2
4258:             02960 FINEH     .BS 1
4259:             02970 FINEV     .BS 1
425A:             02980 NEWFH     .BS 1
425B:             02990 NEWFV     .BS 1
425C:             03000 CHANGE    .BS 1
425D:             03010 ENABLE    .BS 1
```

Strike Force—A Scrolling Game

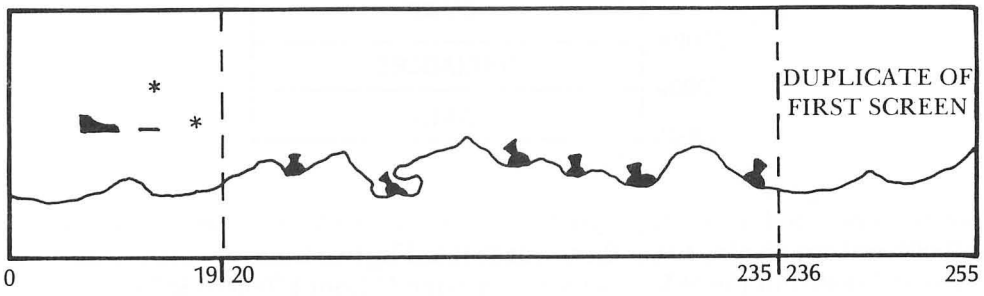
Since horizontally scrolling shoot-'em-up arcade games like *Super Cobra* and *Scrambler* are immensely popular and not difficult to implement on the Atari, we included one called *Strike Force* as an example. The game involves flying an attack ship in a mission of destruction against an enemy attack fleet and their ground installations. The joystick-controlled ship is armed with bombs and lasers. It can fly at two speeds forward but can't reverse direction. Up and down joystick movements control altitude, while pushing the stick forward doubles the speed. The trigger fires the ship's lasers, except when the stick is pushed to the left. That drops bombs.

The continuously scrolling terrain stretches over thirteen screens. The last screen is a duplicate of the first to allow for wraparound. The tan-colored terrain consists of redefined Graphics 1 (ANTIC 6) characters using playfield register #0. The screen data for each of the twenty-two mode lines is 256 bytes or one page of memory. Screen memory requires 5½K.

Four active laser bases and seven missile bases populate the mountainous terrain. These redefined characters reference playfield register #1. They are red in color. While the missile bases don't launch their rockets, the laser bases produce deadly laser fire which should be avoided.

The player's ship, player #0, is double-width and light blue. Its single-width bombs use player #3. The aliens, one green and one red, use players #1 and #2 respectively. Since the ship's laser fire consists of quadruple-width missiles and the aliens use single-width projectiles, we can't combine the missiles to make an extra player. This limits the number of aliens on the screen at any one time to two.

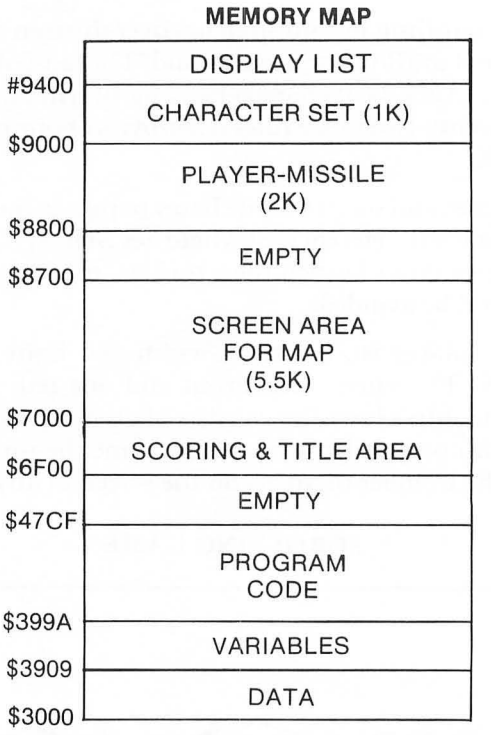
SCROLLING GAME



While a two-prong alien attack can be handled quite skillfully by seasoned players who quickly learn patterns, the aliens in this game are chosen randomly from five different shapes and five different pre-programmed attack patterns. Thus, an alien shape doesn't necessarily correspond to a specific attack pattern. This subtlety makes learning the game difficult. Also, since games should increase in difficulty as they progress, alien firepower increases at 400 points, and again at 2000 points, until the game becomes nearly impossible for the average player.

Memory Layout

Once you have defined the basic design concept of your game, you need to decide where to put your program code, screen memory, display list, character set, and player-missile area. There aren't many constraints to where you put things in the Atari, except that the player-missile area must be on a 2K boundary, and the character set must be on a 1K boundary, and you should avoid the lower portion of memory, especially if DOS is used to load the program. The fact that *Synassembler* resides from $9C00 to $BFFF forces us to locate everything below that area. There-



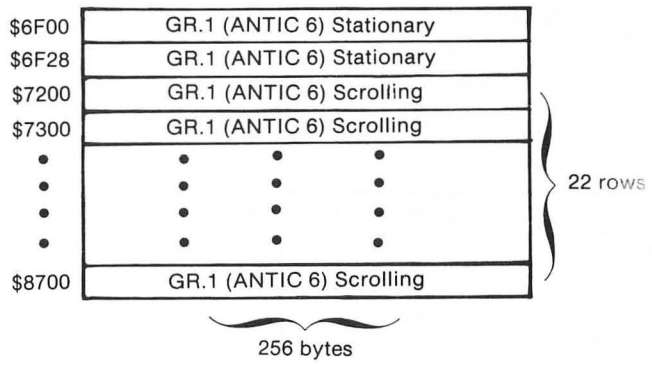
fore, we choose to locate our player-missile area at $8000, the character set above that at $9000 and our display list at the top at $9400. The two lines of scoring information begin at $6F00 and the 5½K of screen area extends from $7000 to $8700.

We assembled our code at $3000, but we could have moved it higher in memory. It really doesn't make any difference in this case since our program code contains the data for the screen, character set, and display list. It wouldn't be any shorter if everything were pushed closer together in memory.

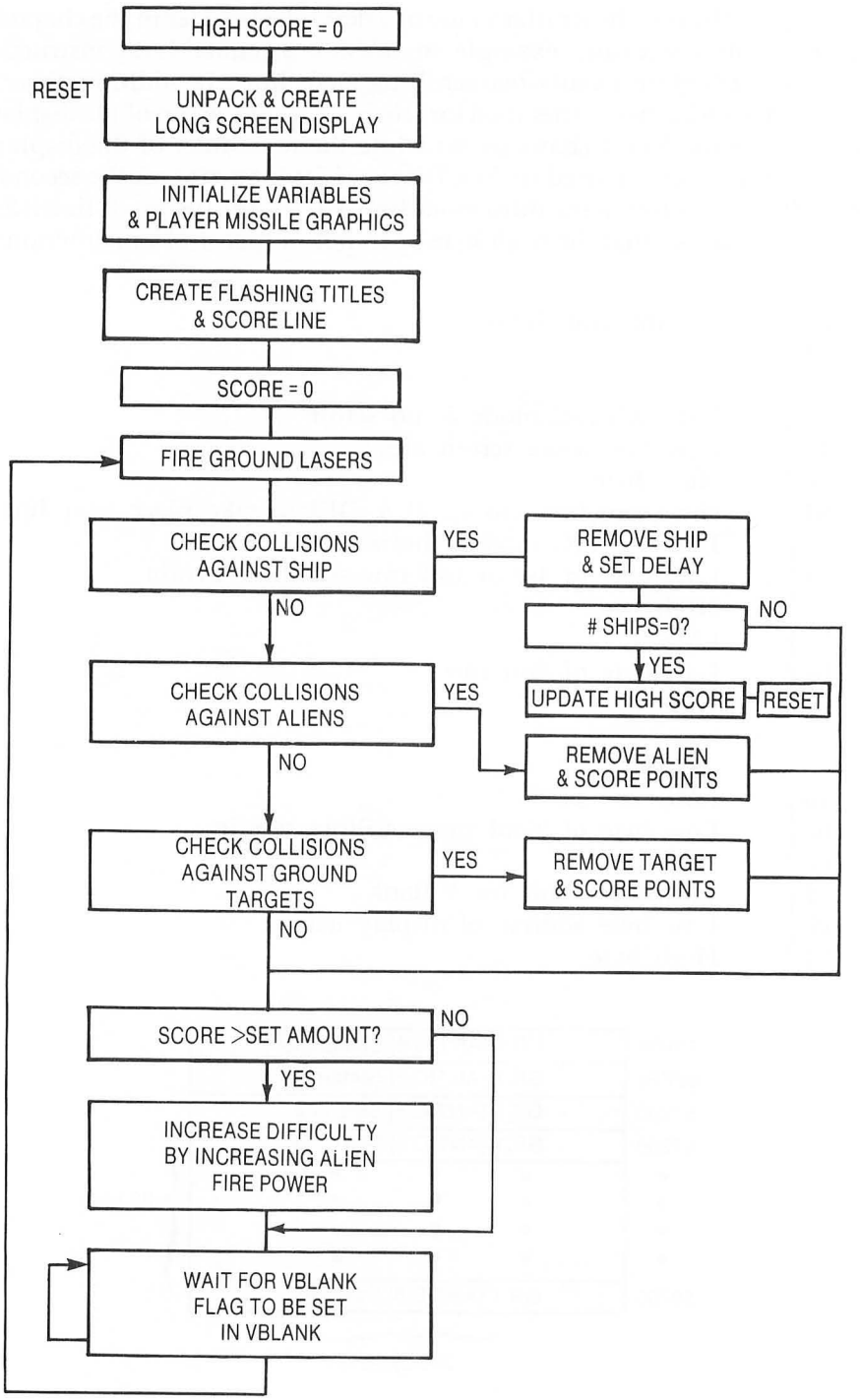
Display List

The display list is quite similar to the one developed earlier in the chapter for the rough horizontal scrolling example in BASIC. Separate LMS instructions are required for each of the twenty-two scrolling mode lines. In addition, there are two stationary ANTIC 6 mode lines used for scoring data at the top of the display. Since these lines use the ROM character set while the remainder of the display uses a custom character set, we need to do a Display List Interrupt on the second line in order for it to take effect in the third mode line. Each scrolling mode line is 256 bytes or one page apart so that the high-byte operands of each LMS instruction are one apart.

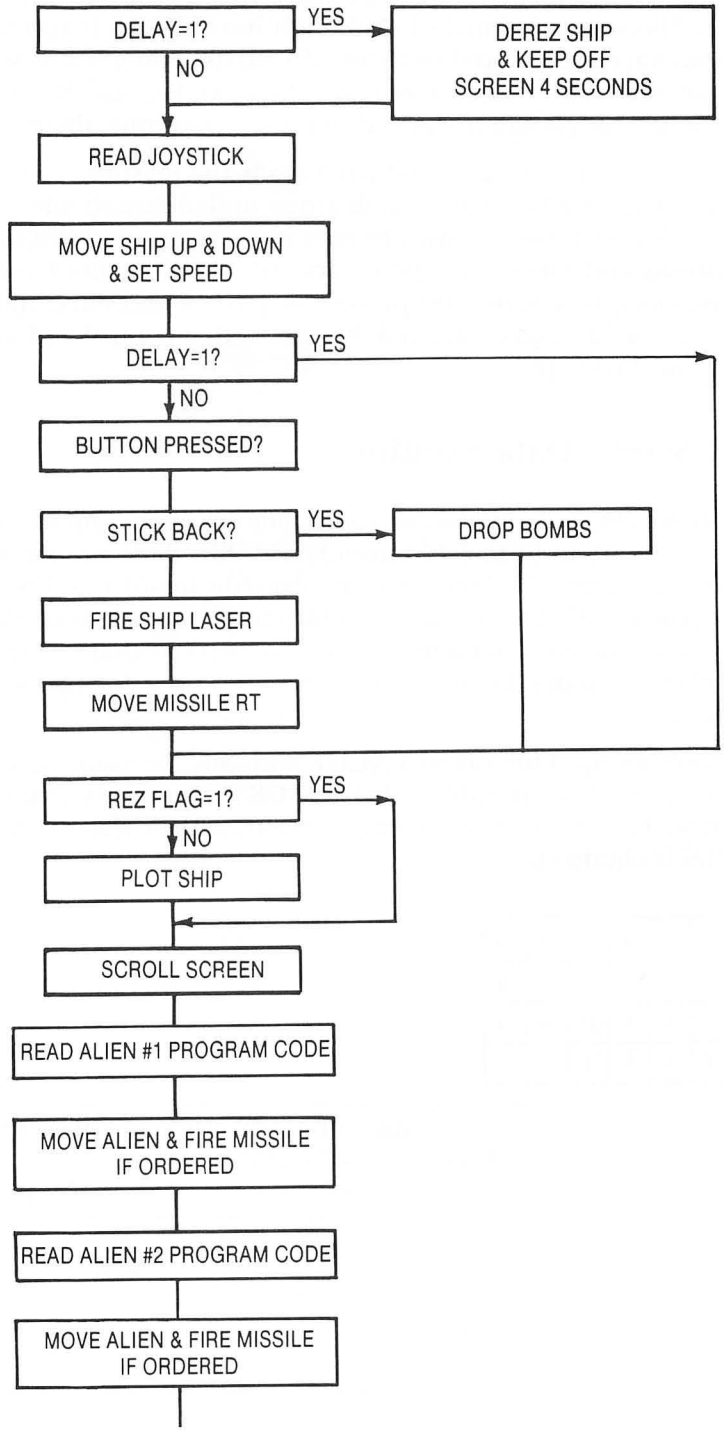
```
$70     8 blank scan lines
70
70
46 )
00 }    LMS ANTIC mode 6 -no scroll
6F )    Low byte score screen area
        High byte
86      2nd score line -no scroll & DLI to take place next line
56 )
00 }    LMS ANTIC mode 6 horiz. scroll
72 )    Low byte of 1st or top row scrolling terrain
        High byte
56 )
00 }    LMS
73 )    Low byte of 2nd row
        High byte
.
.
56 )
00 }    LMS
86 )    Low byte of 22nd row scrolling terrain
        High byte
41 )
00 }    Jump and wait for VBlank
94 )    Low byte address of display list
        High byte
```



Main Loop—Overall Flowchart



VERTICAL BLANK OVERALL FLOWCHART



The overall flowchart of the game is divided into Vertical Blank code and a main code loop outside Vertical Blank. The two sections are synchronized so that the main loop code will execute once, then wait in a tight loop for a flag to be set, signaling that the Vertical Blank code is finished. Although in retrospect it appears that the entire game could have been placed in Deferred VBlank, doing it this way assured that in the event the code became too long, the next Vertical Blank Interrupt wouldn't occur while the program was still in the previous one, thereby crashing.

The Vertical Blank code includes code that reads the joystick, controls all the ship's functions, including lasers and bomb drops, and interprets and executes the programmable code for the two aliens. The code in the main loop checks collisions in all combinations and takes appropriate action. This includes removing shot aliens and ground targets, derezing the player's ship when necessary, and updating the score. In addition, laser base fire, and the checks that control and increase the difficulty, are controlled here.

Unscramble Screen Data Routine

Normally, one would use some sort of homemade scrolling map editor to create the screen data necessary to form a 12-screen world. But these editors, if designed properly, automatically save the data to disk as a data file. In order to allow the reader to type in the required 5½K of screen data without supplying an editor, the data was compacted to a mere 550 bytes. We simply took into account that the upper half of the screen was blank, and that large sections of the lower portion had sequences of similar characters.

Two tables were set up. One called VALUE contains the value of a particular sequence of blocks, and the other table called BLOCKS contains the number of those characters in a row. The small four-row sample illustrated below shows how the data for the two tables is obtained.

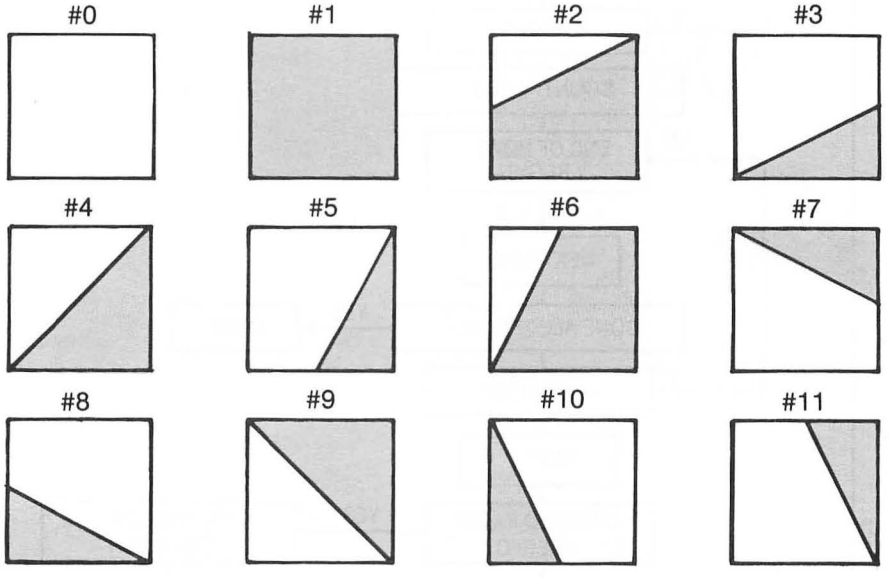
0	0	0	0	4	1	9	0	0	0
0	0	0	4	1	1	1	10	0	0
0	3	2	1	1	1	1	11	0	0
1	1	1	1	1	1	1	1	1	1

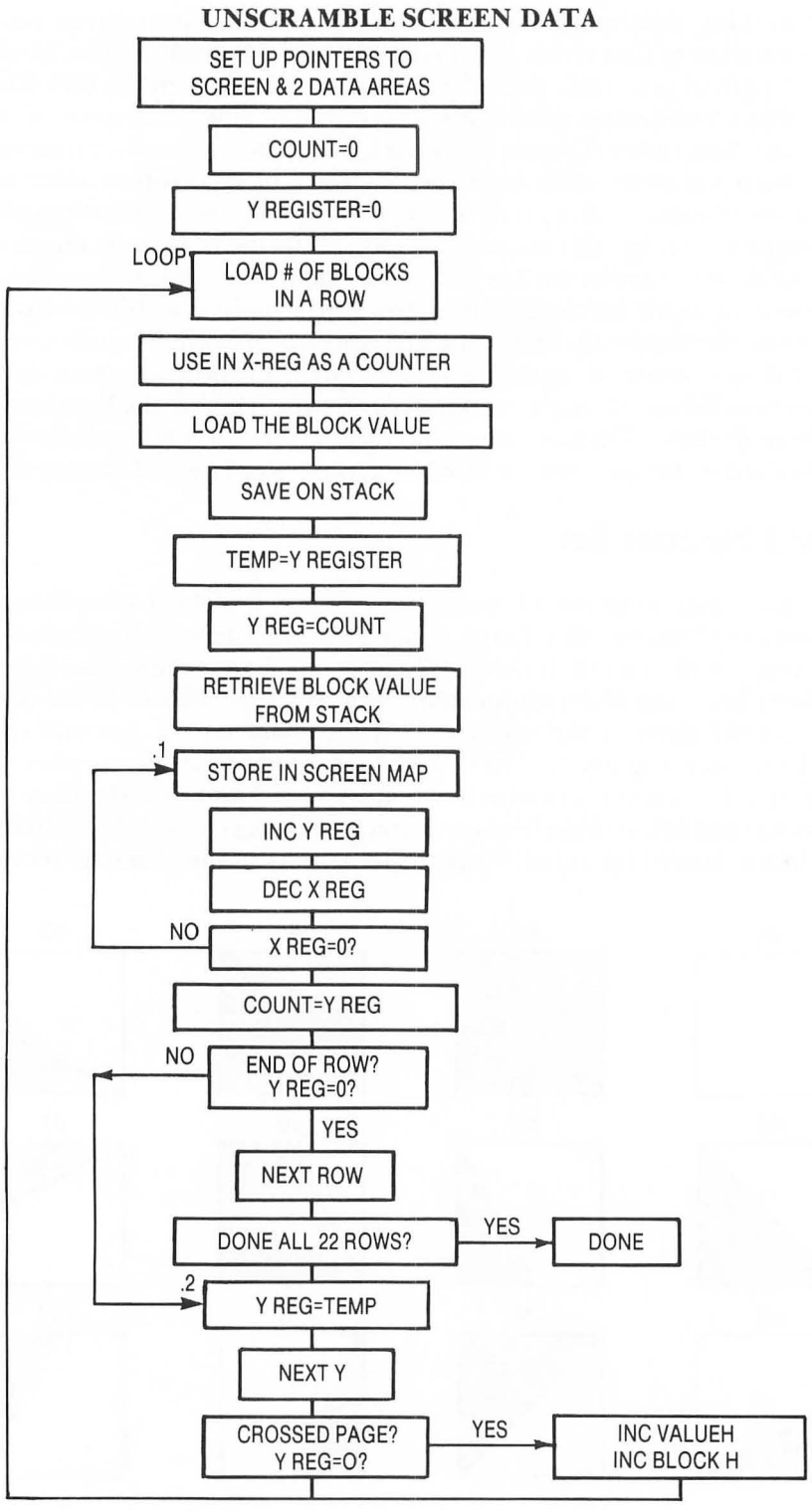
Row #1		Row #2	
Value	# Blocks in a row	Value	# Blocks in a row
0	4	0	3
4	1	4	1
1	1	1	3
9	1	10	1
0	3	0	2

The unpacking routine that places the character data into screen memory is extremely sensitive to data errors in which the length of a single row isn't exactly 256 bytes, the length of one mode line. If a mistake occurs, data in the following rows isn't just offset, but it often doesn't even appear. The routine assumes that it will finish storing the number of repeat character bytes at exactly the same time that the Y register, which has meanwhile been keeping track of its position along the row, reaches the end (zero). At that point it jumps to the next row. Unfortunately, the Y register counter is being incremented in a loop while the number of characters in a row is being decremented in the X register. For example, if the number of blocks in a row were one too many for the 256-block row, the Y register would have passed the zero point and the test for the end of the row would be missed. The row wouldn't be incremented and the rest of the data would overwrite previous character data on the same mode line. While I thought of fixing it by testing whether the Y register became zero while in the loop, I'm not sure which case is worse, knowing that you made a mistake in the data for that row, or observing a completely weird display afterward.

Custom Character Set

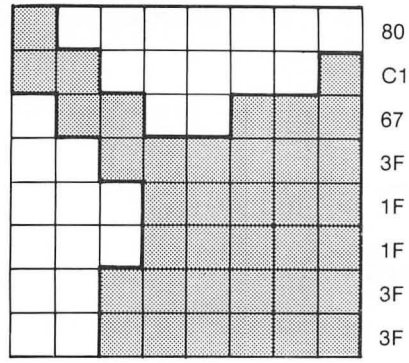
The display uses a custom character set composed of twelve combinations of sloping terrain. Character #0 is blank. Since these are internal characters 0-11, they use color register #0 to set their color. We chose tan. The rockets, laser base, and its moving laser beam are also custom characters. Since we wanted to use a different color, we placed them in the upper half of the character set. Internal characters 64-127 refer to color register #1. The two high bits of the character number select the color register. The rockets are internal characters #63 and #64, the laser bases are characters #60 and #61, and the beam is character #62. We chose red primarily so that the laser beam showed up as red. Since the remainder of the character set was of no



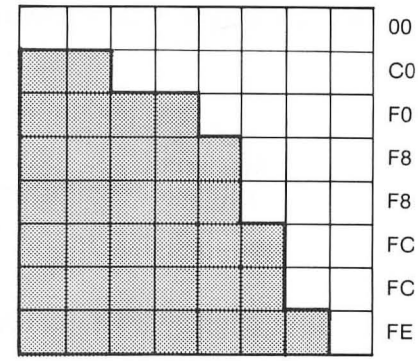

UNSCRAMBLE SCREEN DATA
SET UP POINTERS TO SCREEN & 2 DATA AREAS
COUNT=0
Y REGISTER=0
LOOP
LOAD # OF BLOCKS IN A ROW
USE IN X-REG AS A COUNTER
LOAD THE BLOCK VALUE
SAVE ON STACK
TEMP=Y REGISTER
Y REG=COUNT
RETRIEVE BLOCK VALUE FROM STACK
.1
STORE IN SCREEN MAP
INC Y REG
DEC X REG
NO
X REG=0?
COUNT=Y REG
NO
END OF ROW? Y REG=0?
YES
NEXT ROW
DONE ALL 22 ROWS?
YES
DONE
.2
Y REG=TEMP
NEXT Y
CROSSED PAGE? Y REG=O?
YES
INC VALUEH INC BLOCK H

interest, it wasn't copied from ROM and then modified. Instead, the character data was copied from tables directly into the specified positions in character set memory.

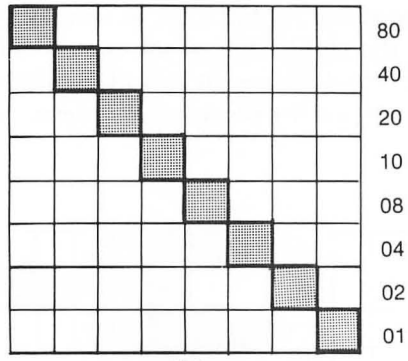
However, since we did need the letters for the GAME OVER message which scrolls across the playfield at the conclusion of the game, the data for these letters are stored as characters $65-$71. They too, appear in red. All of these tables occupy only 192 bytes of program memory.



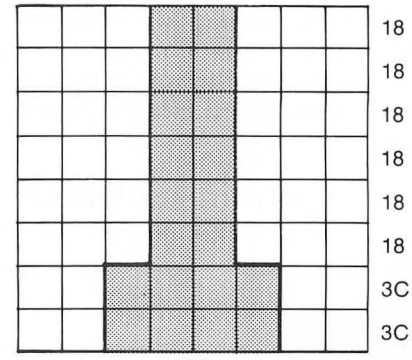
#60



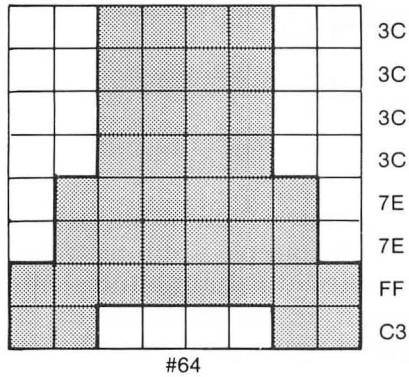
#61



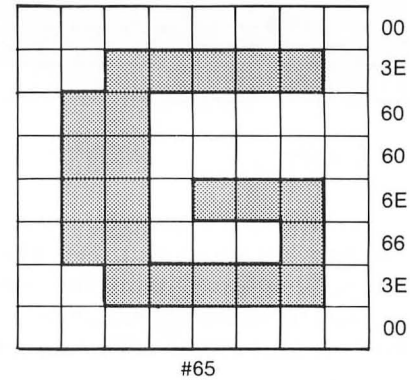
#62



#63



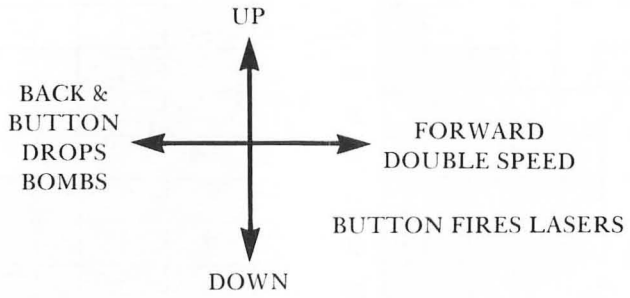
#64



#65

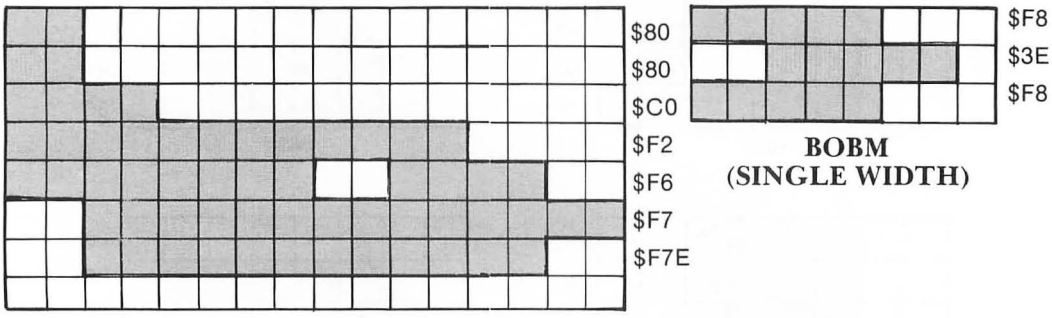
Ship Operation

The joystick-controlled spaceship, while stationary on the horizontal axis during scrolling, can be moved vertically to adjust altitude. Pushing forward or to the left on the stick doubles the speed and consequently the rate of scrolling and horizontal speed of the attacking aliens. The latter is necessary to maintain the alien's position above the faster scrolling terrain. When the stick is pushed forward, the variable SPEED is set to one, otherwise it remains zero. There are also two different engine



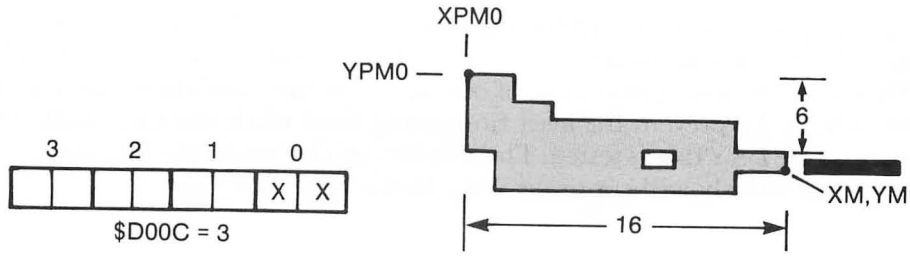
sounds depending on the speed. Keeping the ship stationary in the horizontal direction was supposed to keep the programming simpler. Still, it wouldn't have altered the code much if the ship were free to move in that axis, as long as it was restrained from approaching too closely to the right edge of the screen. This is necessary to prevent the bombs from falling beyond the screen boundary. Introducing this motion might make an interesting exercise for those who understand the program.

SHIP (DOUBLE WIDTH)



Ship's Laser

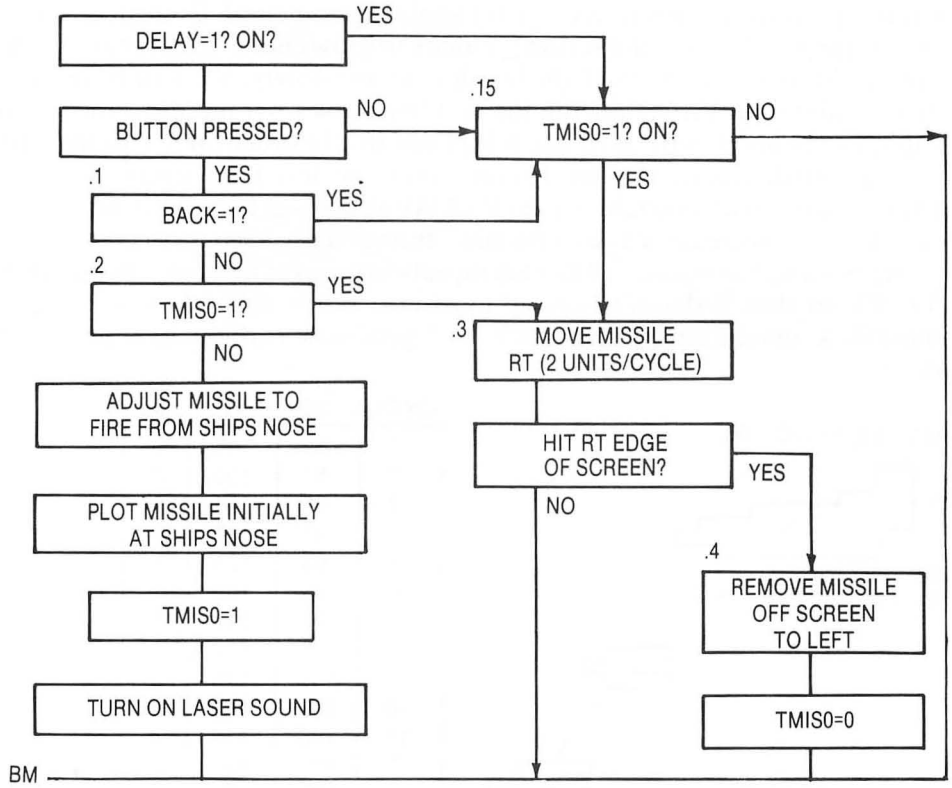
Pressing the joystick trigger activates the ship's lasers. This works in any of the neutral or forward stick positions. The laser is a quadruple-width missile #0. The size can be set independently of the normal-size alien missiles by setting both of the lower two bits in SIZEM ($D00C = 3). This hardware register is divided into bit pairs for each of the four missiles. The missile shape is just one pixel high by two pixels wide.



Sets Missile 0 = Quadruple width
Reset normal width

Only one ship's missile or laser beam can be on the screen at any one time. To prevent it from refiring before it either strikes its target or exits screen right, a flag called TMIS0 is set. TMIS0 = 0 the first time through the routine so that it adjusts the missile's position to fire initially from the ship's nose and plots it there. It then resets the flag to 1 and turns on the laser fire sound timer. During subsequent cycles through the routine, the TMIS0 flag causes it to branch to the code that moves the

SHIP MISSILE OR LASER



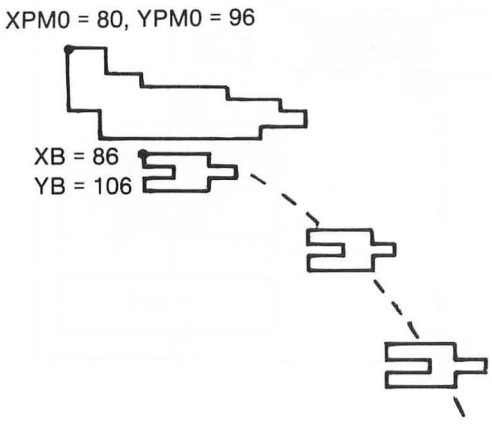
beam 2 units/cycle to the right. The position is tested against the screen boundary on the right side. If it hits it, the missile is removed and placed offscreen to the far left and TMIS0 is set to zero. Collisions, of course, cause the same effect, but they are tested elsewhere. To prevent the laser from being fired while the ship is offscreen after a derez, a DELAY flag is tested. The missile can't be fired if the flag is set, but a missile fired previously will continue along its track.

Bomb Drop

In a realistic bomb drop, the bomb arcs slowly at first and then accelerates rapidly downward until it falls almost entirely vertically. You can implement such a drop on the Atari with only a minimal knowledge of physics. Gravity or acceleration only acts on the bomb in the Y direction. While the velocity in the Y direction increases with time, the velocity in the X direction remains constant, if we neglect air resistance. An object that has a velocity in a particular direction will move in that direction. Its new position is equal to its old position plus its change in position during that period (velocity). We can summarize this as follows:

```
VY = VY + GRAVITY      YB = YB + VY
VX = CONSTANT          XB = XB + VX
```

Choosing a realistic acceleration value is largely experimental. Even an acceleration of +1/frame would cause the vertical velocity to grow enormous over as little as 15 animation frames (¼ second). If the bomb is to arc slowly, VY will have to be somewhat smaller than VX = 2 during the first few frames, yet not grow too much larger later, or the bomb will drop like a lead weight. In order for VY to increase slowly every fourth frame, the acceleration must be less than unity. A clever approach is to increment a variable called VTEMP and divide by 4 each time. It will take four cycles to increase VY by one unit/frame. This keeps the bomb from accelerating too fast, but it still will fall too rapidly from great heights. The solution is to clip VY so that it doesn't become too high. While the values are largely experimental, a maximum value of VY = 3 produces realistic-looking bomb trajectories.

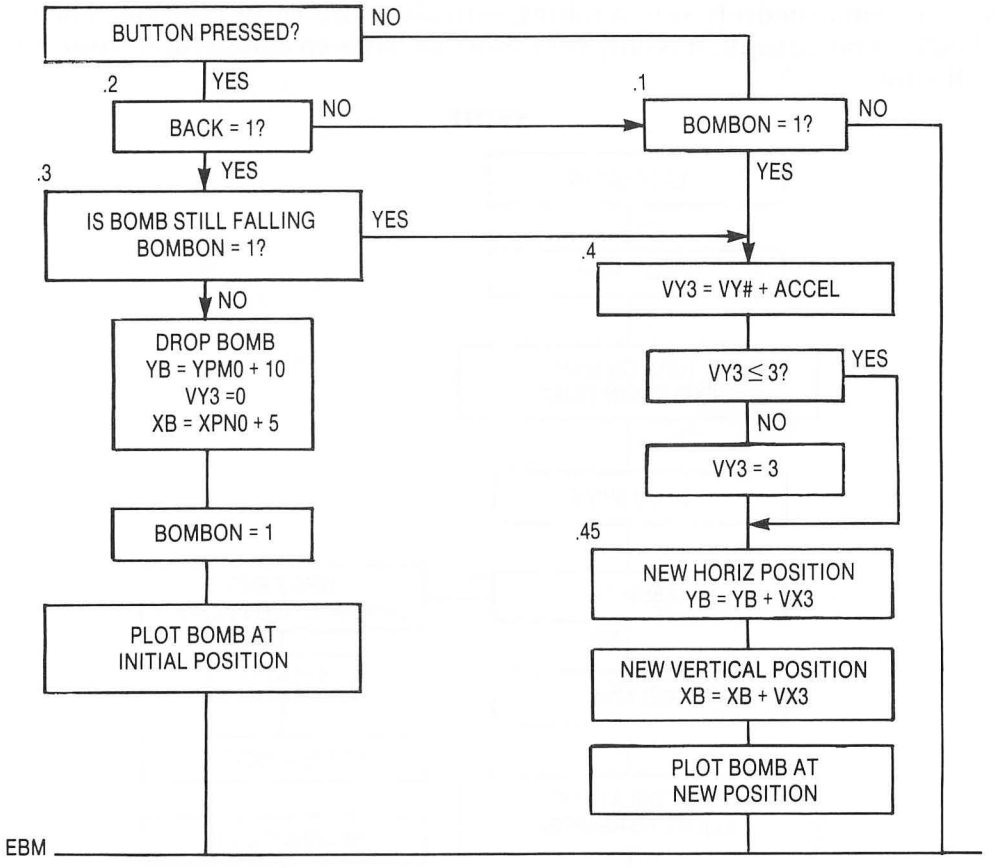


CYCLE	XB	YB	VY
1	85	106	0
2	87	106	0
3	89	106	0
4	91	107	1
5	93	108	1
6	95	109	1
7	97	110	1
8	99	112	2
9	101	114	2
10	103	116	2
11	105	118	2
12	107	121	3
13	109	124	3

The bomb subroutine uses a flag called BOMBON to determine if it should plot the bomb initially directly beneath the plane, or accelerate and move it as it falls. The bomb must begin its descent from the center of our plane because bomb bays are located at the plane's center of gravity. Therefore, the bomb needs repositioning from the ship's coordinates XPM0, YPM0 at the tip of the tail. YB = YPM0+10 and XB = XPM0+5. The bomb is plotted there, and the BOMBON flag is set to 1.

When the bomb subroutine is entered on subsequent frames, it branches to the bomb falling code where the bomb's velocity and position for each direction is calculated. It is then plotted in that position. Of course, the bomb is removed during the collision test if it strikes either the ground or any of the laser base or missile targets, but that occurs elsewhere in the main line code. The BOMBON flag is reset to zero there, so that another bomb can be dropped if the trigger is pressed while the stick is pushed back or to the left.

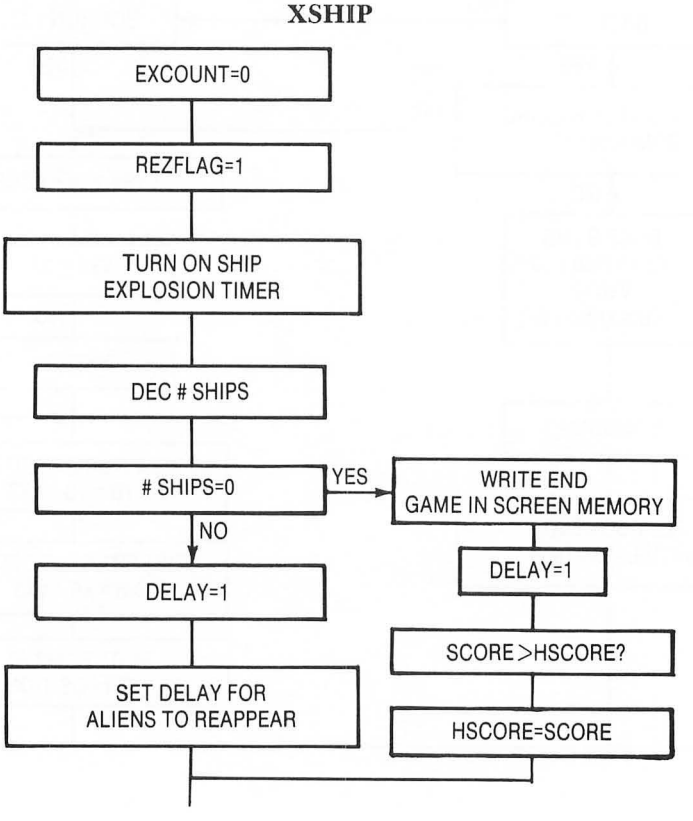
BOMB DROP FLOW CHART



Ship's Explosion and Delay

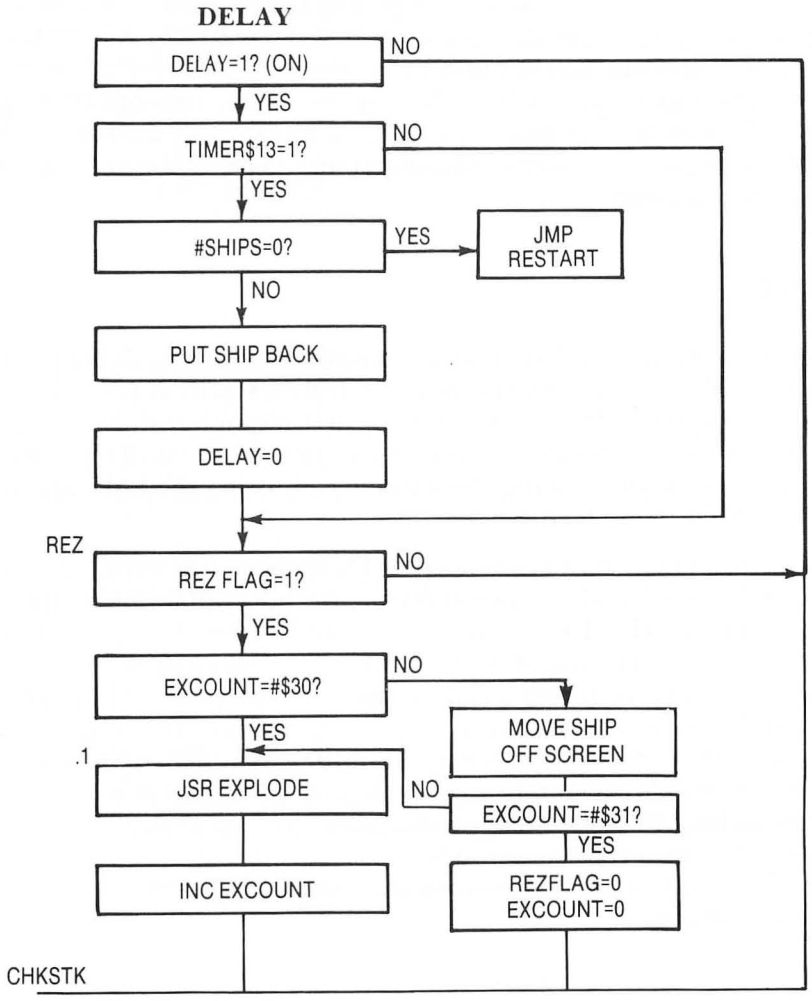
The player's ship explodes upon collision with the terrain, an alien, or enemy fire from either the laser bases or alien craft. Although there are many methods to explode a ship, we choose to use the deresolution method employed earlier in the *Space War* game in Chapter 5. The appearance is of a ship that is slowly disintegrating. As you recall, it uses a random number generator to degrade a duplicate of the ship's shape. It is a fairly complicated routine that uses a series of AND and ORA instructions to control the image's rate of degradation so that the random flickering of the individual pixels lasts at least forty-eight cycles. We aren't going to explain the routine again, so we suggest you look at it in the last section of Chapter 5.

Since their should be a several second rest between the ship's deresolution and its reappearance to battle once again, many of the subroutines that operate on a per cycle basis should be shut off during this period. These include the start of another enemy attack once they clear the screen, and the ability to fire lasers and drop bombs once your ship is killed. While it might be easier to set one delay flag during the explosion and branch past all of the code, you need to be selective in what is shut down, otherwise laser beams and falling bombs might vanish in mid-flight and the screen would suddenly stop scrolling with aliens frozen in position. The game should go on. Afterall, it is only your ship that has been destroyed; everyone else is still alive.



We need to set several flags in the XSHIP subroutine that is called whenever the ship collides with anything. Setting REZFLAG = 1 turns on the actual explosion subroutine. Setting DELAY = 1 starts the four second delay. In addition, this lengthens the timer delays, NDELAY1 and NDELAY2, that control when the next set of aliens reappear. If we are out of ships, the words "Game Over" are written into screen memory just beyond the right edge of the screen.

There is a section of code at the beginning of the Vertical Blank Interrupt routine that actually monitors all of these flags so that the ship is derezed first and then removed from the screen until the four second delay ends. It keeps track of the timer at location $13. This location is incremented roughly every four seconds. When it is



NOTES: Delay Flag—Keeps Ship Off Screen
RezFlag—Says Explosion On (1)
Excount—Keeps Track of Explosion Frames
=0 Explosion Off

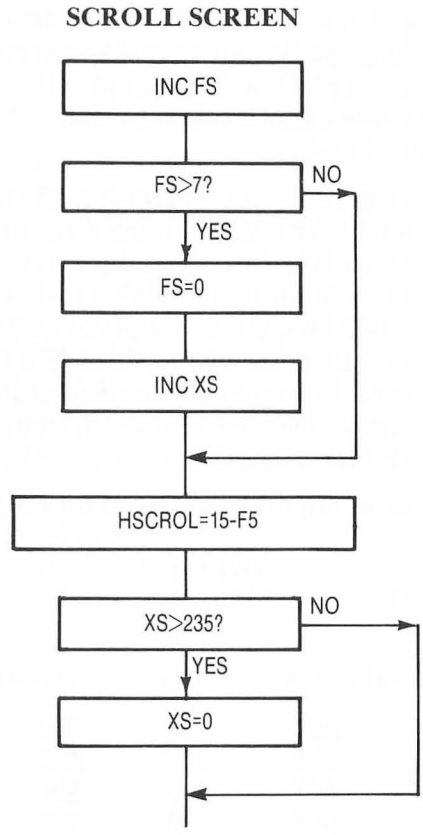
non-zero, the routine checks if any ships are left, and if so, puts the ship back and resets the delay timer to zero. Obviously, if we are out of ships, the game is over, and it is time to jump to the very beginning of the game code.

Long before the timer at $13 ever becomes non-zero, the routine determines where it is in the derez cycle via a counter called EXCOUNT. The explosion subroutine increments EXCOUNT after each cycle. When EXCOUNT reaches 48 cycles, the ship is virtually disintegrated. This is done one cycle earlier than the cycle that resets REZFLAG = 0. If at least one of the ship's pixels is in contact with the playfield, a collision value will be returned even though we just ordered the ship offscreen. The computer updates it's collision registers before we have time to move our ship. If we don't delay setting REZFLAG = 0 by one cycle, our collision test may detect a bogus collision and the player loses another ship. The ship is then moved off the screen. This is done one cycle earlier than the cycle that resets REZFLAG = 0 because if at least one of the ship's pixels is in contact with the playfield, a collision value will be returned even though we just ordered the ship off screen. The computer updates it's collision registers before we have time to move our ship. If we don't delay setting REZFLAG = 0 by one cycle, our collision test may detect a bogus collision and the player loses another ship.

Scrolling

The screen scrolls leftward at a constant rate of one color clock per second except when the control stick is pushed to the right. It then scrolls at two color clocks per second so that it appears that the ship is flying at twice the speed. To scroll the screen smoothly you need to alternate between adjusting the fine scroll register over eight color clocks, and rough scrolling the screen by adjusting the LMS operands of all twenty-two ANTIC 6 mode lines.

In our case, we increment a variable called FS, short for fine scroll, each cycle. This variable goes from 0-7, and is backwards from the horizontal fine scrolling register HSCROL at $D404. HSCROL is initially set at 15 when FS equals 0 and counts down as FS increases. Thus, HSCROL = 15-FS. The fine scroll register is reset every eight clock cycles and the rough scroll variable XS is incremented. When the screen has been rough scrolled 235 times, it is time to stop and reset the rough scroll register XS back to zero. If we try to go further, for example to XS = 236, ANTIC will fetch the first nineteen bytes of that scan line correctly, but the twentieth will be data for the following scan line. This obviously produces a faulty display that becomes worse the further we move into the wraparound zone. This end zone should be an exact duplicate of the first screen so that when we jump from XS=235 to XS=0, the screen will remain unchanged.



Programmable Aliens

It usually requires an extensive amount of program logic to achieve a variety of enemy patterns within a game. Some games, like this one, use preset patterns that are independent of the player's action, while others react to evasive techniques used by the player. Obviously, the computer's reaction to the player's actions produces more challenging games, but if the programmer isn't careful he may design a game in which it is impossible to survive.

For example, in this game aliens follow some sort of zig-zag pattern, change their speed, and shoot in predetermined directions as they close on your position. It would be quite easy to program them to home in on your position with their guns or just on your vertical position for eventual collision. There would be only two possible results: you would either be able to shoot them easily if they matched your vertical position quickly for they would always be directly in your line of fire; or they would home in on you at the last split second and you would have no chance of survival. Likewise, you could never evade their guns, if they were programmed to shoot with uncanny accuracy. Perhaps a better method might be to make random variations on a predetermined path.

We chose a slightly different tack to solve the problem of easy programmability, while still offering enough variations to make learning the alien attack patterns difficult. We actually developed five different patterns for each alien player, and to make it easy to change wrote a scheduling routine that bases all enemy actions on an internal timer and a set of tables.

The first three bytes in the table contain the X1, Y1 starting position of the alien, followed by the value NDELAY1, which is the delay in cycles before the next alien appears after the current one is killed or exits the screen. Each group of six bytes that follows is an instruction containing the high-and-low-byte time to read the next instruction (TIME1L,TIME1H), the alien velocity (VX1,VY1), whether to shoot or not (SHOOT1), and the direction of the shot (DIR1). We are limited to eight different instructions for each programmable alien because five different patterns are stored in one 256-byte page of memory. Thus, each complete program occupies 6*8 + 3 = 51 bytes of memory. We have set aside two blocks of memory, one for each player.

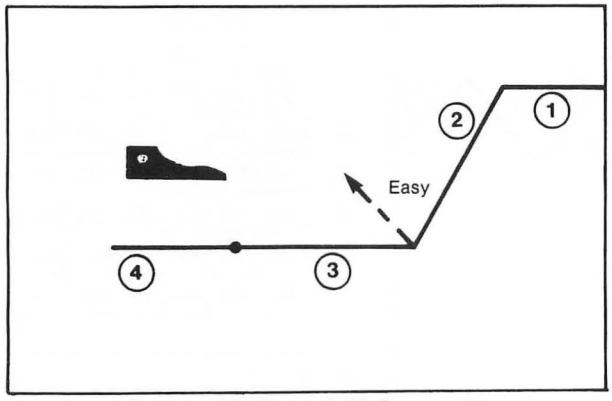
If you look at the data in our table ENEMY1 for the 0th shape, it is as follows:

X1	Y1	NDELAY1
$D0	$50	$30

TIME1L	TIME1H	VX1	VY1	SHOOT	DIR
$28	$00	$FF	$00	$00	$00
$50	$00	$FF	$01	$00	$00
$70	$00	$FF	$00	$01	$07
$FF	$00	$FF	$00	$00	$00

If you match it against the diagram below, you will see that the alien enters the screen at X=$D0, Y=$50. These are player-missile screen coordinates and are completely independent of the scrolling playfield. The value NDELAY1 = $30 means that there will be a delay of forty-eight screen cycles between one programmable alien using player #2 leaving the screen, and the next one entering. The alien begins moving leftward in step with the scrolling terrain below until the internal timer (TIMER1) reaches $28. It then reads in the next six-byte instruction. This instruction tells it to begin moving diagonally downward. VX1=$FF and VY1=$01. It also obtains a new timer value of $50 so that the next instruction isn't read until its internal timer reaches $50. At that time VY1 = 0 and the alien continues its path along the horizontal axis. Its gun is turned on, and it shoots in a direction of 7, up and to the left. It reads its last instruction at TIME1=$70 and shuts off its guns. It will continue moving horizontally until it exits screen left.

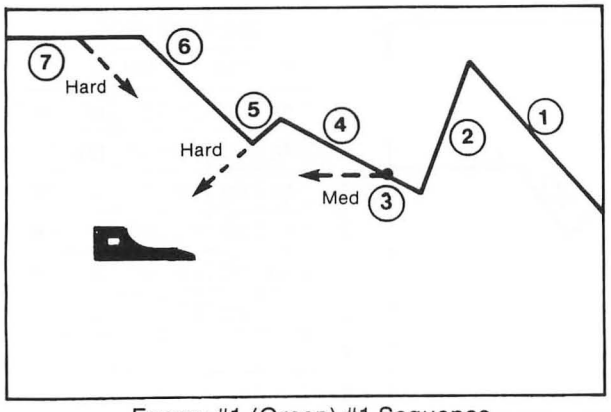
There is a delay timer, TDELAY1, that prevents a new alien from appearing immediately after the first. The value of NDELAY1 that was looked up in the tables by the prior alien is transferred to TDELAY1 after that alien exits the screen. If TDELAY1 is positive when it enters the routine that reads the programmable alien code, it branches past it and just decrements this timer once each cycle. There is a secondary flag called ONSCRN1 that is set to one during alien initialization after the



Enemy #1 (Green) #0 Sequence

X	Y	NDELAY
D0	50	30

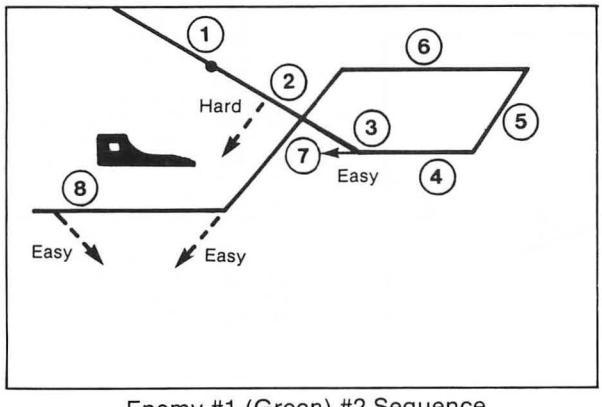
	TL	TH	VX	VY	SHOOT	DIR
1	28	00	FF	00	00	00
2	50	00	FE	01	00	00
3	70	00	FF	00	01	07
4	FF	00	FF	00	00	00
5						
6						
7						
8						



Enemy #1 (Green) #1 Sequence

X	Y	NDELAY
D0	75	40

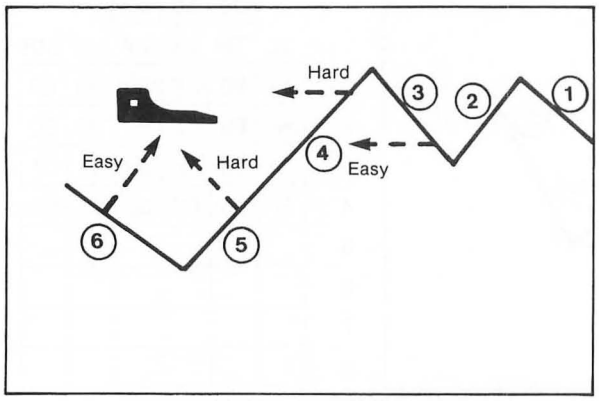
	TL	TH	VX	VY	SHOOT	DIR
1	30	00	FF	FF	00	00
2	55	00	00	01	00	00
3	65	00	FF	FF	00	00
4	75	00	FF	FF	01	06
5	80	00	FF	01	01	05
6	98	00	FF	FF	00	00
7	FF	00	00	FF	01	03
8						



Enemy #1 (Green) #2 Sequence

X	Y	NDELAY
50	32	25

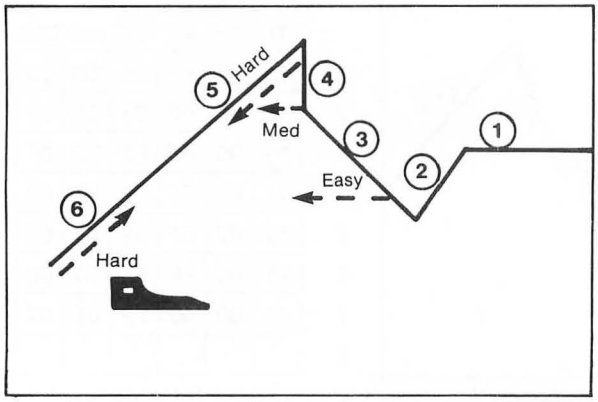
	TL	TH	VX	VY	SHOOT	DIR
1	15	00	01	01	00	00
2	30	00	01	01	01	05
3	40	00	01	00	01	06
4	53	00	01	00	00	00
5	70	00	01	FF	00	00
6	80	00	FF	00	00	00
7	A8	00	FF	01	01	05
8	FF	01	FF	00	01	03



Enemy #1 (Green) #3 Sequence

	X	Y	NDELAY
	D0	70	35

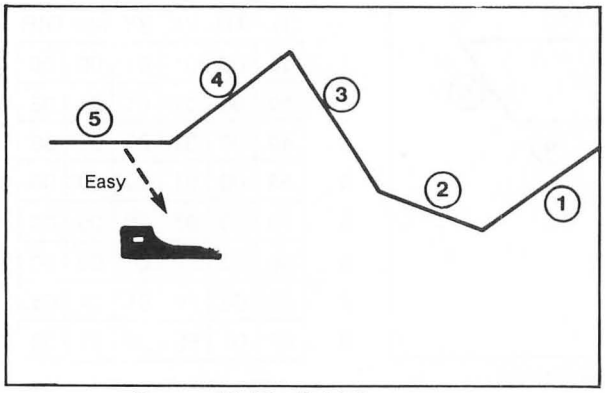
	TL	TH	VX	VY	SHOOT	DIR
1	20	00	FF	FF	00	00
2	40	00	FF	01	00	00
3	60	00	FF	FF	01	06
4	80	00	FF	01	01	06
5	98	00	FF	01	01	07
6	FF	00	FF	FF	01	01
7						
8						



Enemy #1 (Green) #4 Sequence

	X	Y	NDELAY
	D0	65	15

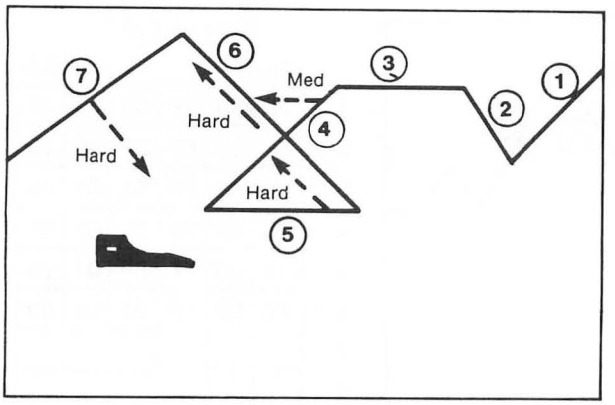
	TL	TH	VX	VY	SHOOT	DIR
1	25	00	FF	00	00	00
2	40	00	FF	01	00	00
3	60	00	FF	FF	01	06
4	80	00	00	FF	01	06
5	A0	00	FF	01	01	05
6	FF	00	FF	00	01	01
7						
8						



Enemy #2 (Red) #0 Sequence

	X	Y	NDELAY
	D0	40	50

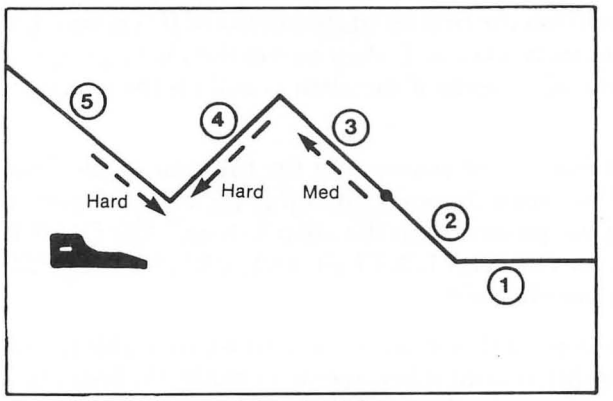
	TL	TH	VX	VY	SHOOT	DIR
1	38	00	FF	01	00	00
2	50	00	FF	FF	01	06
3	68	00	FF	FF	00	00
4	78	00	FF	01	00	00
5	90	00	FF	00	01	03
6	FF	00	FF			
7						
8						



Enemy #2 (Red) #1 Sequence

	X	Y	NDELAY
	D0	35	20

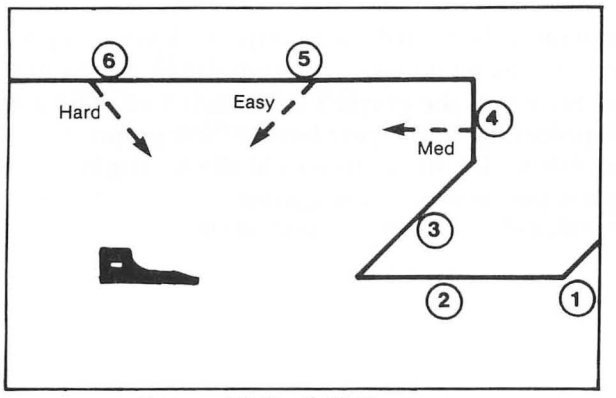
	TL	TH	VX	VY	SHOOT	DIR
1	18	00	FF	01	00	00
2	30	00	FF	FF	00	00
3	40	00	FF	00	00	00
4	60	00	FF	01	01	06
5	70	00	01	00	01	07
6	80	00	FF	FF	01	07
7	FF	00	FF	01	01	03
8						



Enemy #2 (Red) #2 Sequence

	X	Y	NDELAY
	D0	80	20

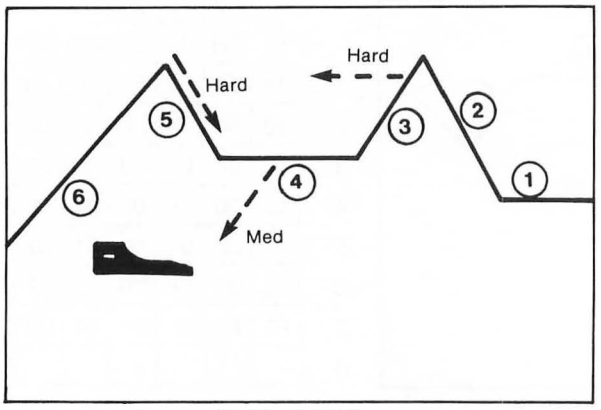
	TL	TH	VX	VY	SHOOT	DIR
1	30	00	FF	00	00	00
2	50	00	FF	FF	00	00
3	60	00	FF	FF	01	07
4	78	00	FF	01	01	05
5	FF	00	FF	FF	01	03
6						
7						
8						



Enemy #2 (Red) #3 Sequence

	X	Y	NDELAY
	D0	55	20

	TL	TH	VX	VY	SHOOT	DIR
1	15	00	FF	01	00	00
2	35	00	FE	00	00	00
3	45	00	01	FF	00	00
4	55	00	FF	00	01	06
5	70	00	FF	00	00	00
6	95	00	FF	00	01	05
7	FF	00	FF	00	01	03
8						



Enemy #2 (Red) #4 Sequence

	X	Y	NDELAY
	D0	65	20

	TL	TH	VX	VY	SHOOT	DIR
1	20	00	FF	00	00	00
2	38	00	FF	FF	00	00
3	50	00	FF	01	01	06
4	68	00	FF	00	01	05
5	78	00	FF	FF	01	03
6	FF	00	FF	01	00	00
7						
8						

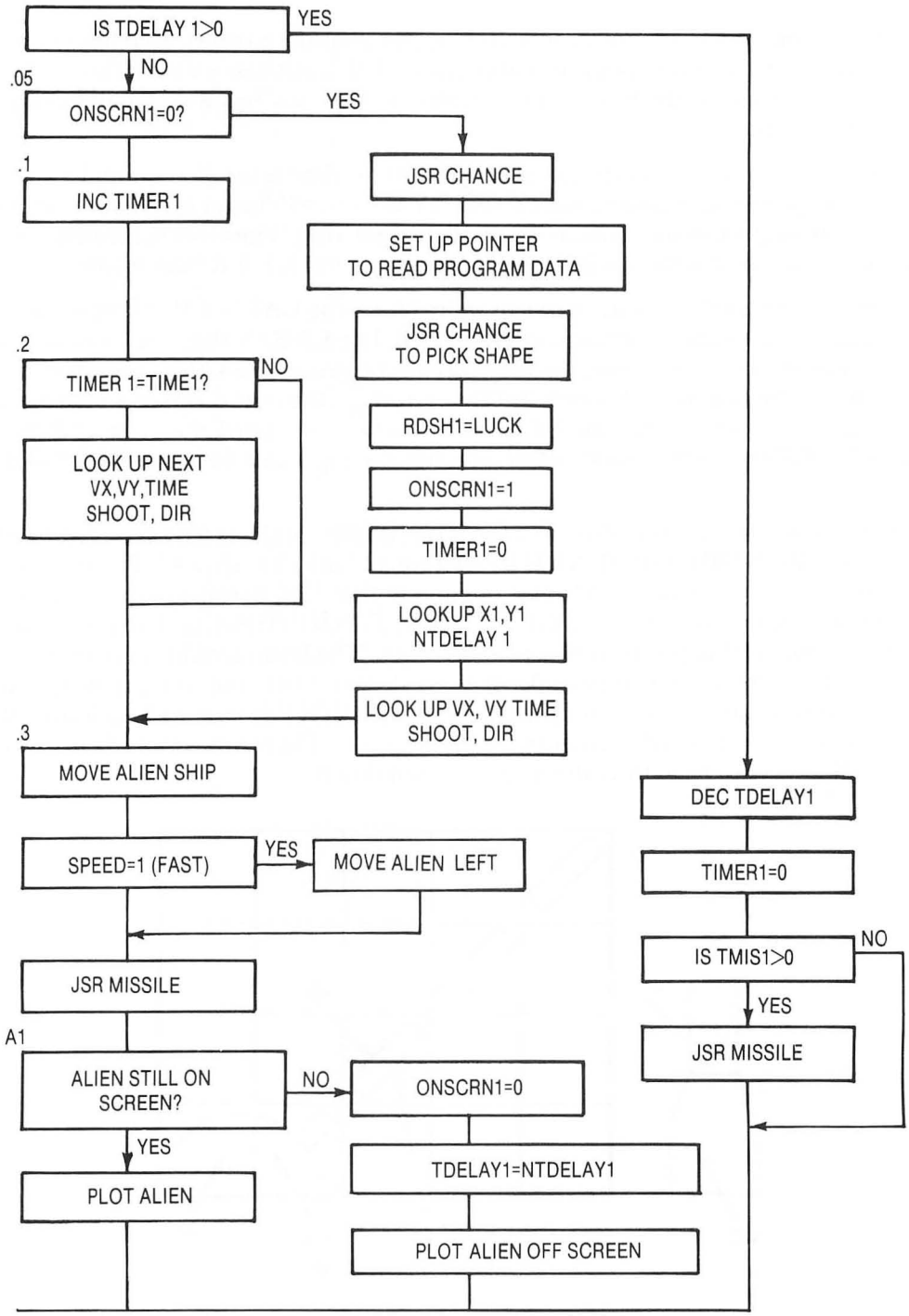
delay has ended. Once this flag is set, the program compares its timer, TIMER1, to that of TIME1 which it obtained from the first set of instructions. If it is equal, the program looks up the next set of instructions. It then moves the alien, jumps to a separate subroutine to fire the missile, checks if the alien is still on the screen, and finally plots the alien.

During initialization, one of five sets of pointers to the programmable data is chosen randomly. Then, one of five alien shapes is randomly picked. This prevents the player from predetermining the pattern from the alien's shape. The ONSCRN flag is then set and the initial values for X1,Y1, NTDELAY1, VX1, VY1, TIMEL1, TIMEH1, SHOOT1, and DIR1 are obtained.

The obvious advantage of developing this routine was that we were able to create an attack pattern, test it, then modify it until it became sufficiently challenging. Of course, when we did this, we hadn't put in the CHANCE subroutine. Therefore, it was easier to control which programmable alien code we were working with. Since the X-register determines which program is loaded, it is fairly easy to skip the CHANCE subroutine and load the X-register with a value from 0 to 4.

The game turned out to be much more difficult than anticipated because the level of alien firepower was too intense. It was relatively easy to cut the firepower at the beginning of the game, and then raise it as the player's score increased. The game shifts to intermediate level at 400 points, then to expert level at 2000 points. Most of the shoot flags in the programmable tables are set to zero at the beginning of the game and then restored as the game progresses. Consequently, it was quite easy to develop a game that increases in difficulty with the player's skill.

ALIEN #1 CODE


IS TDELAY 1>0
YES
NO
.05
ONSCRN1=0?
YES
.1
INC TIMER 1
JSR CHANCE
SET UP POINTER
TO READ PROGRAM DATA
JSR CHANCE
TO PICK SHAPE
.2
TIMER 1=TIME1?
NO
RDSH1=LUCK
LOOK UP NEXT
VX,VY,TIME
SHOOT, DIR
ONSCRN1=1
TIMER1=0
LOOKUP X1,Y1
NTDELAY 1
LOOK UP VX, VY TIME
SHOOT, DIR
.3
MOVE ALIEN SHIP
DEC TDELAY1
SPEED=1 (FAST)
YES
MOVE ALIEN LEFT
TIMER1=0
IS TMIS1>0
NO
JSR MISSILE
YES
A1
JSR MISSILE
ALIEN STILL ON
SCREEN?
NO
ONSCRN1=0
YES
TDELAY1=NTDELAY1
PLOT ALIEN
PLOT ALIEN OFF SCREEN

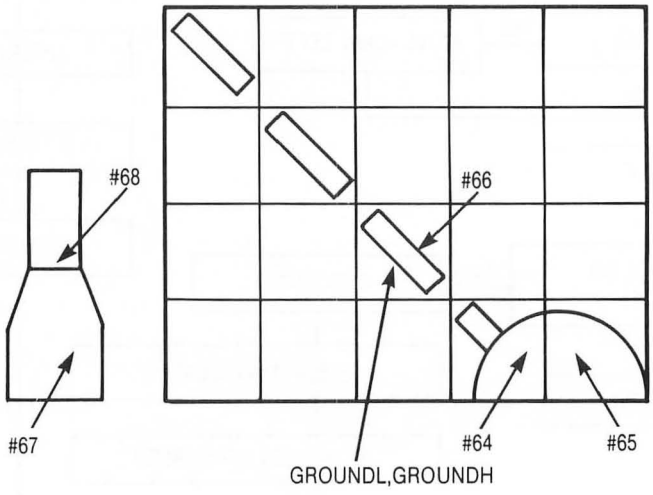
Ground Lasers

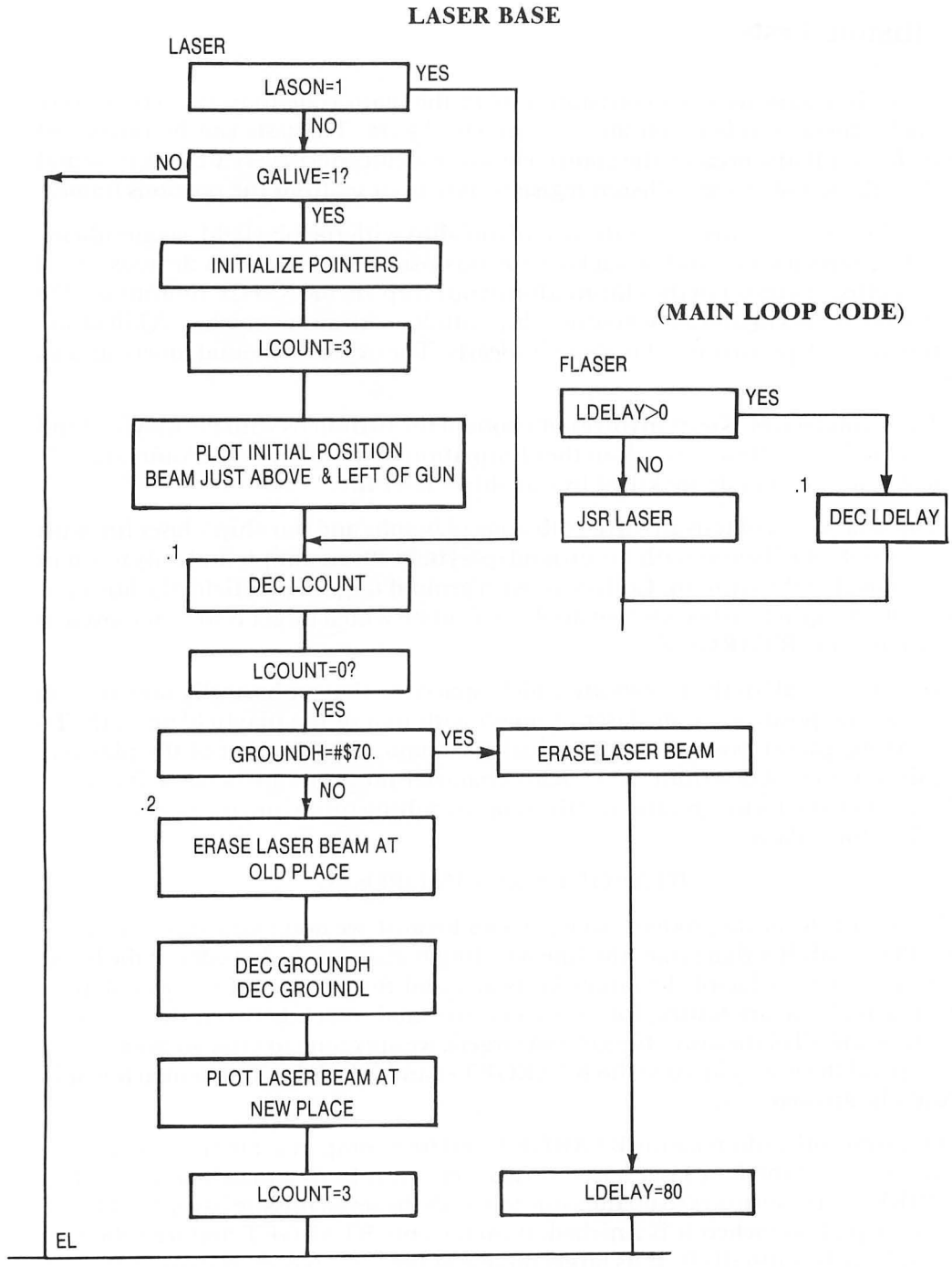
The four ground lasers use animated character graphics to move their laser beams. The segment of the beam uses internal character #62. Each base uses two flags to keep track of the status of the base and two timers to slow the beam down and delay it between firings.

The main loop code performs a simple test to determine if it should call the subroutine LASER. A timer called LDELAY is set to #$50 after each shot to insure that it rests slightly more than one second before refiring. The timer decrements once each cycle, so eventually it will reach zero and call the LASER subroutine.

Since it is possible to destroy one or more bases, the GALIVE flag keeps track of active bases. Bases are alive when GALIVE = 1. The LASON flag is used to prevent the routine from reinitializing the beam after it begins its track from just above and to the left of the laser base. When LASON = 1, it skips the initialization and proceeds with the movement each time LCOUNT reaches zero until the beam, which is plotted at GROUNDL,GROUNDH, reaches the top mode line at GROUNDH = #$70.

The beam, which is traveling at a forty-five degree angle, is first erased at its old position GROUNDL,GROUNDH by writing a blank character #0. The new position is one character to the left and one mode line (256 bytes) lower in memory. Therefore, the new position is at GROUNDL-1,GROUNDH-1, and we plot internal character #62 in this position of screen memory. The beam would move much too rapidly if it were moved every cycle. We can slow it down and move it every third cycle by our usual countdown timer method. LCOUNT is reset to 3 each time the beam is moved and then decremented with each cycle. The beam is moved only when LCOUNT reaches zero, thus slowing the beam down.




LASER BASE
LASER
LASON=1
YES
NO
NO
GALIVE=1?
YES
INITIALIZE POINTERS
LCOUNT=3
PLOT INITIAL POSITION
BEAM JUST ABOVE & LEFT OF GUN
.1
DEC LCOUNT
LCOUNT=0?
YES
GROUNDH=#$70.
YES
ERASE LASER BEAM
NO
.2
ERASE LASER BEAM AT
OLD PLACE
DEC GROUNDH
DEC GROUNDL
PLOT LASER BEAM AT
NEW PLACE
LCOUNT=3
LDELAY=80
EL
(MAIN LOOP CODE)
FLASER
LDELAY>0
YES
NO
.1
JSR LASER
DEC LDELAY

Collision Tests

There is a long series of collision tests in the main code loop that cover every possible interaction between any two screen objects. The tests can be performed outside the VBlank because the main code only executes once every VBlank cycle and will do the test after the collision registers have been set from the previous frame.

The first few tests involve collisions of our ship with the playfield, enemy aliens, and their missiles. Ground-based laser fire is considered playfield in the tests. In all cases, collisions result in the elimination of our ship via the XSHIP subroutine. We only score points in the case where our ship collides with an enemy ship. A kill of any nature is worth points even if it costs you dearly. The explosion sound timers are also set here.

The second series of tests involve collisions of the two aliens with the playfield and our ship's laser. All cases result in the elimination of the alien ship. Points are only scored if the alien craft are killed by our ship's laser fire.

The final series of tests involve collisions of bombs and our ship's laser fire with the playfield. Collisions with the ground (playfield #0) are simple and only result in the removal of the weapon. Collisions with ground targets (playfield #1), however, are more complicated because we need to calculate which target is to be removed in the subroutine RTARGET.

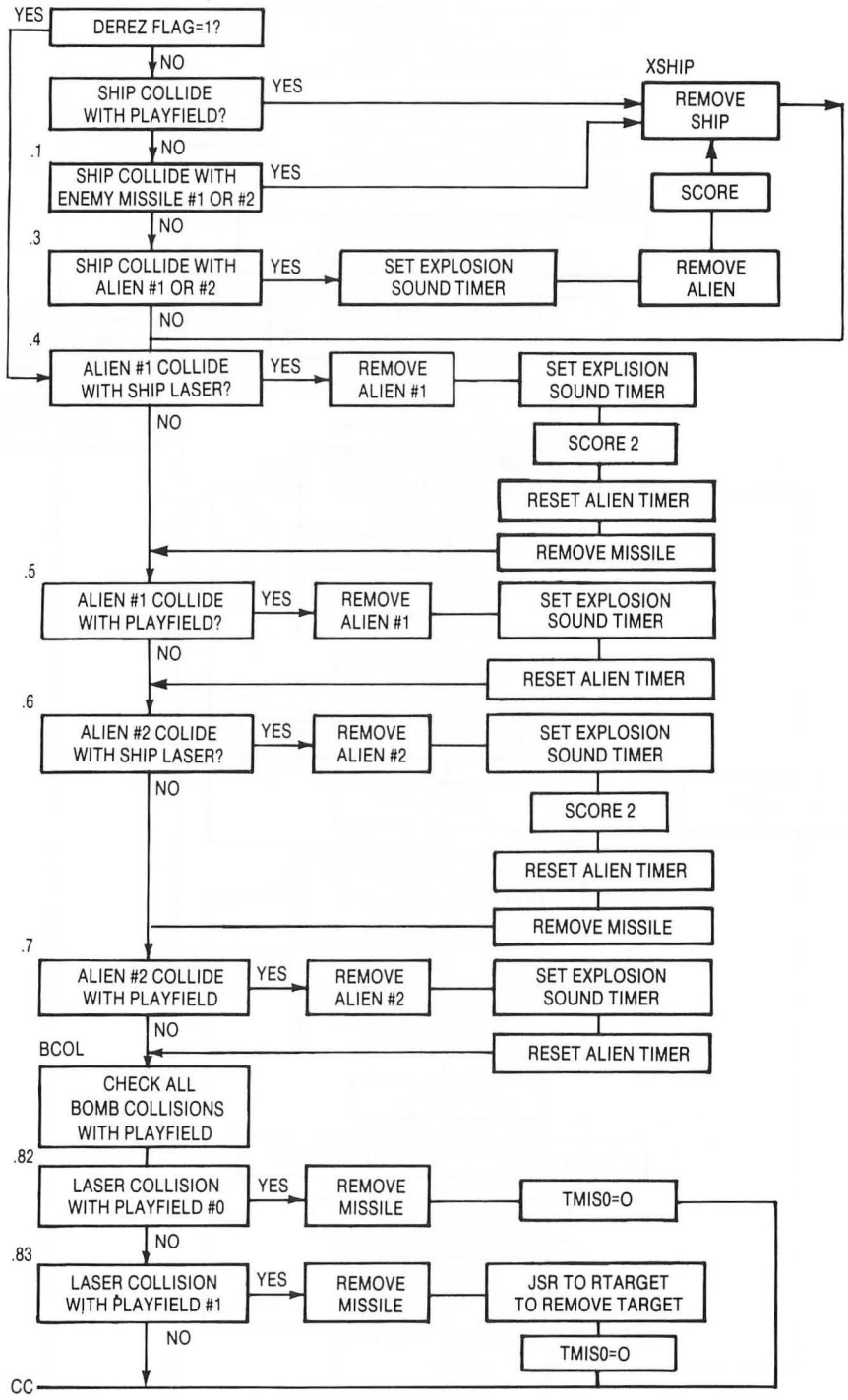
Fortunately, all of the targets are widely spaced so that we basically only need to compare the position of our laser or bomb with that of the playfield beneath. To convert the player position of the missile or bomb (POS) to that of the playfield requires us to first determine how many character positions are between it and the playfield's left screen edge, and add the rough scrolling offset into the playfield map, XS. The formula is:

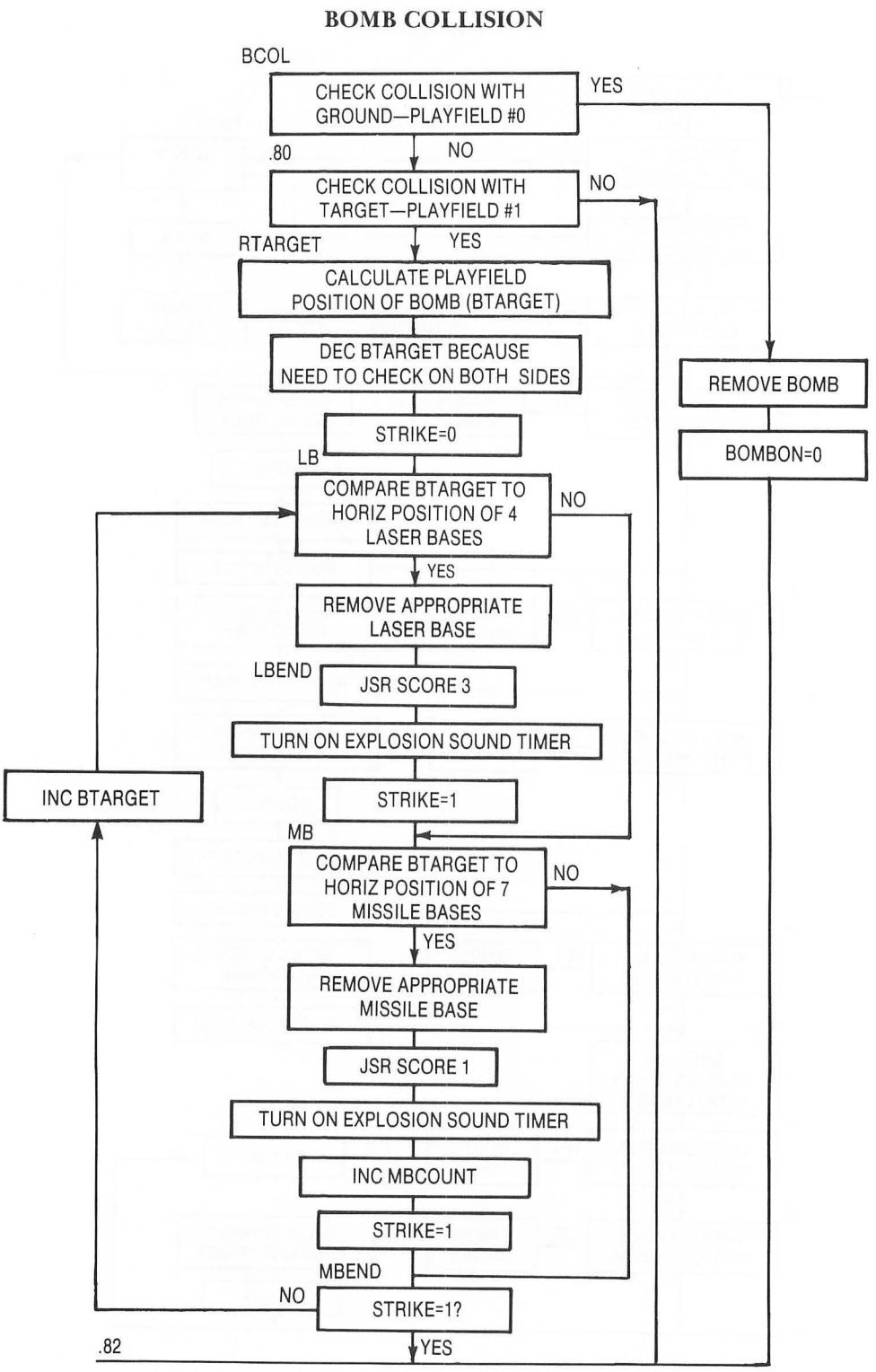
$$BTARGET = XS + INT(POS/8)$$

Using this formula produces inaccuracies because we don't take into consideration the possibility that either the fine scrolling register or the left edge of the bomb may clip the far edge of the target in its arc, and thus throw off our calculation. Obviously, if we are testing for an exact horizontal match between the weapon's position and all of the available ground targets, we are going to miss occasionally. If we expand the test to include the BTARGET-1 and BTARGET+1, a match would always be assured.

Our subroutine decrements BTARGET and first compares it to horizontal positions of each of the four laser bases. It removes it if it finds a match and sets a flag STRIKE = 1. It continues to do the same test with the seven missile bases. If STRIKE doesn't equal one when it is finished, it increments BTARGET and tries the tests again. It will eventually find its target on one of the three passes, remove it, trip the explosion sound time, score some points, and exit the subroutine.

COLLISIONS


YES
DEREZ FLAG=1?
NO
SHIP COLLIDE WITH PLAYFIELD?
YES
XSHIP
REMOVE SHIP
.1
NO
SHIP COLLIDE WITH ENEMY MISSILE #1 OR #2
YES
SCORE
.3
NO
SHIP COLLIDE WITH ALIEN #1 OR #2
YES
SET EXPLOSION SOUND TIMER
REMOVE ALIEN
NO
.4
ALIEN #1 COLLIDE WITH SHIP LASER?
YES
REMOVE ALIEN #1
SET EXPLISION SOUND TIMER
NO
SCORE 2
RESET ALIEN TIMER
REMOVE MISSILE
.5
ALIEN #1 COLLIDE WITH PLAYFIELD?
YES
REMOVE ALIEN #1
SET EXPLOSION SOUND TIMER
NO
RESET ALIEN TIMER
.6
ALIEN #2 COLIDE WITH SHIP LASER?
YES
REMOVE ALIEN #2
SET EXPLOSION SOUND TIMER
NO
SCORE 2
RESET ALIEN TIMER
REMOVE MISSILE
.7
ALIEN #2 COLLIDE WITH PLAYFIELD
YES
REMOVE ALIEN #2
SET EXPLOSION SOUND TIMER
NO
BCOL
RESET ALIEN TIMER
CHECK ALL BOMB COLLISIONS WITH PLAYFIELD
.82
LASER COLLISION WITH PLAYFIELD #0
YES
REMOVE MISSILE
TMIS0=O
NO
.83
LASER COLLISION WITH PLAYFIELD #1
YES
REMOVE MISSILE
JSR TO RTARGET TO REMOVE TARGET
NO
TMIS0=O
CC

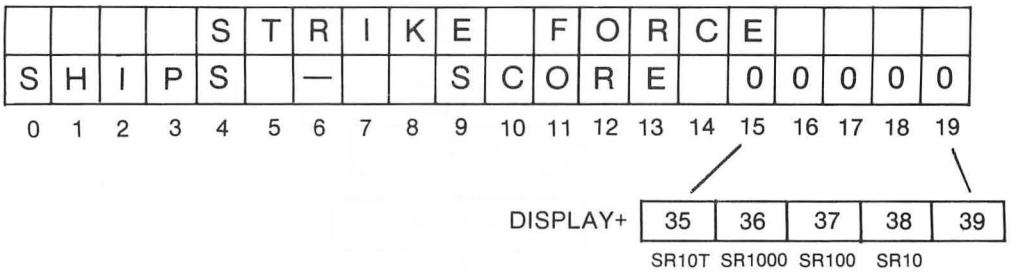

BOMB COLLISION
BCOL
CHECK COLLISION WITH GROUND—PLAYFIELD #0
YES
.80
NO
CHECK COLLISION WITH TARGET—PLAYFIELD #1
NO
RTARGET
YES
CALCULATE PLAYFIELD POSITION OF BOMB (BTARGET)
DEC BTARGET BECAUSE NEED TO CHECK ON BOTH SIDES
REMOVE BOMB
BOMBON=0
STRIKE=0
LB
COMPARE BTARGET TO HORIZ POSITION OF 4 LASER BASES
NO
YES
REMOVE APPROPRIATE LASER BASE
LBEND
JSR SCORE 3
TURN ON EXPLOSION SOUND TIMER
INC BTARGET
STRIKE=1
MB
COMPARE BTARGET TO HORIZ POSITION OF 7 MISSILE BASES
NO
YES
REMOVE APPROPRIATE MISSILE BASE
JSR SCORE 1
TURN ON EXPLOSION SOUND TIMER
INC MBCOUNT
STRIKE=1
MBEND
NO
STRIKE=1?
.82
YES

Scoring

The scoring subroutine differs little from those of many of the others used in this book. It and others keep track of the different decimal digits separately so that they don't have to convert an internal hexadecimal score to decimal. There are separate counters for the tens digit, hundreds digit, thousands digit, etc. Carrys are performed into the next higher digit whenever one or more of these digits exceeds a value of nine. The actual positions of each of these counters in the score line is shown below.

The scoring values for each target in the game are worth differing point values. Missile bases are worth 10 points, alien ships 20 points, and laser bases 30 points. Fairly high scores can be achieved by players since there are an unlimited number of aliens, and missile bases are replenished after all seven have been destroyed.

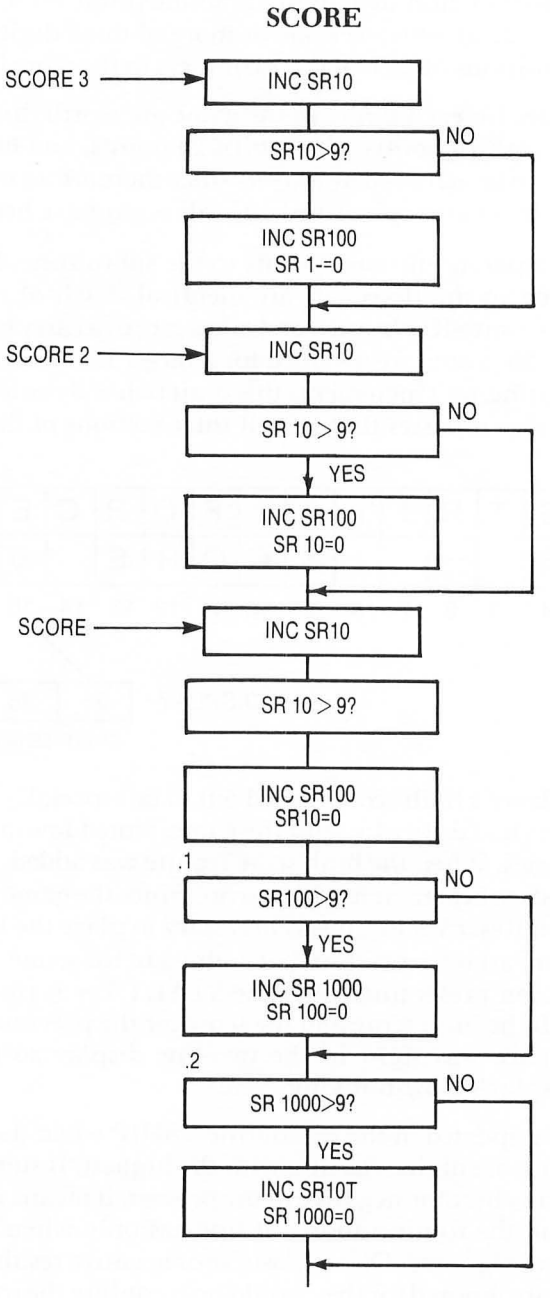
There are three separate entrance points to the subroutine. Notice that the first three blocks of each in the flowchart are identical. Each of these small sections increments the tens digit SR10 by one, and takes care of a carry to the hundreds digit if necessary. When 20 points are awarded for a target it enters at SCORE2, a lower point in the subroutine, and increments this digit twice. By entering at SCORE3, 30 points are scored since it passes through all three sections of the same code.

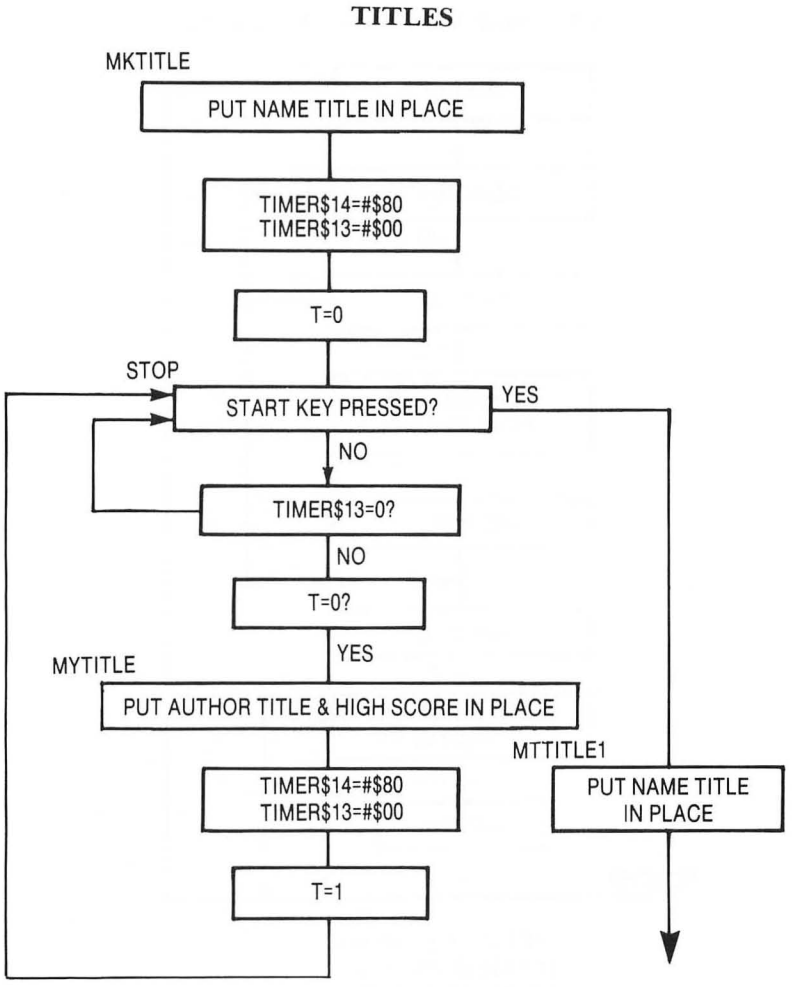


Retaining the player's high score turned out to be especially important in *Strike Force*. Many players had difficulty with the game, scored low, and refused to play it after two or three tries. When the high score feature was added, they would play for much longer periods to beat their last high score. Since the game resets automatically through the System Reset Vector, it was necessary to place the initialization for the high score digits or variables prior to that address in the game. The score variables for the last game aren't reset until after the START key is pressed. This makes it possible to see both the high score and the score for the previous game while in the "attack mode." Space was tight in the two-line display so that the two scores alternate in the display's slow-blinking cycle.

The high score is updated in the subroutine XSHIP when the game is over. Each of the current high score digits, starting with the highest, is subtracted from each of the score digits. If any become negative in the process, it means we haven't reached a new high score and the routine aborts. It updates only when it detects a positive result. Zero results are ignored. Once a positive or negative result is obtained, the rest of the lower digits are ignored for they would only confuse the routine. For example,

if SCORE were 00630, and HISCORE were 00580, it would get a positive number in the hundreds digit. This means we have a new high score. If we were to continue the test we would get a negative number in the tens digit. This would mean we don't have a new high score. Since only the first positve or negative result is meaningful, it is best to ignore the remaining digits.



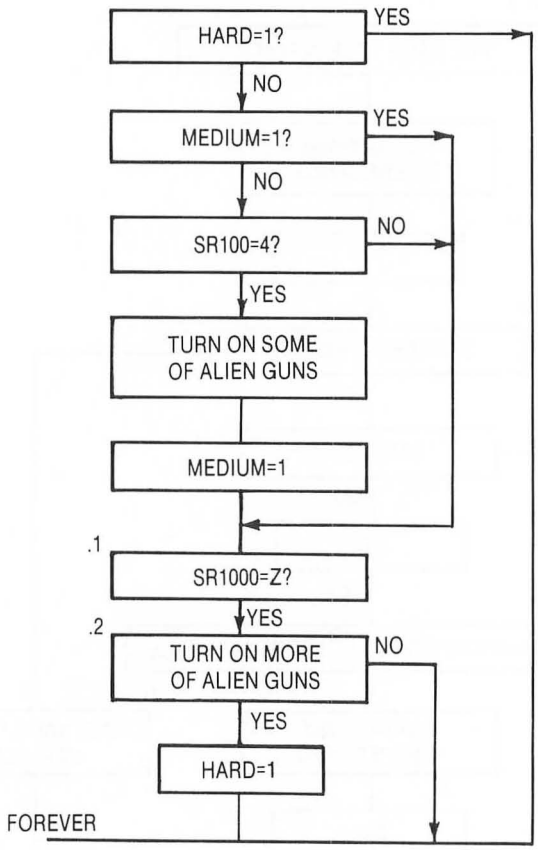


Sound

The game uses all four sound channels. Channel one is used for the laser fire sound, channel two for the short explosion sounds of the aliens and ground targets, channel three for the engine sound, and channel four for the sound of the ship's longer explosion. Four separate channels are required so that one sound doesn't interfere with another in the event that several occur simultaneously.

The sound subroutine resides and operates in the VBlank routine. They are essentially always on but require setting a positive value in a flag to trip them. For example, the laser sound can be tripped by setting SLTIME = 1. Each of the routines, which are described more fully in the sound section in chapter eight, use internal countdown timers to gradually lower the tone and volume. The engine sound, on the other hand, just produces a continuous sound using some distortion. The frequency is changed to a higher pitch when the ship speeds up.

CHANGE DIFFICULTY LEVELS



MEDIUM @ 400 pts
HARD @ 2000 pts
MEDIUM=0 (OFF)

```
                00015 *STRIKE FORCE-COPYRIGHT 1984 - BY JEFFREY STANTON
                00020 *PART 1 SCROLLING GAME
                00025          .OR $4000
                00030          .TF "D:SCROLL.OBJ"
                00035 *ZERO PAGE EQUATES
00F0:           00040 SHPL     .EQ $F0
00F1:           00045 SHPH     .EQ $F1
00F2:           00050 SHPML    .EQ $F2
00F3:           00055 SHPMH    .EQ $F3
00F4:           00060 SHPMOL   .EQ $F4
00F5:           00065 SHPMOH   .EQ $F5
00F6:           00070 GROUNDL  .EQ $F6
00F7:           00075 GROUNDH  .EQ $F7
00F8:           00080 VALUEL   .EQ $F8
00F9:           00085 VALUEH   .EQ $F9
00FA:           00090 BLOCKL   .EQ $FA
00FB:           00095 BLOCKH   .EQ $FB
00FC:           00100 PMADR    .EQ $FC
```

```
00FC:          00105 CHSET    .EQ $FC
00FE:          00110 CHADR    .EQ $FE
00F8:          00115 E1L      .EQ $F8
00F9:          00120 E1H      .EQ $F9
00FA:          00125 E2L      .EQ $FA
00FB:          00130 E2H      .EQ $FB
00F6:          00135 MBZL     .EQ $F6
00F7:          00140 MBZH     .EQ $F7
               00145 *LOCATION EQUATES
7000:          00150 SCREEN   .EQ $7000 ;ADR OF MAP
6900:          00155 INFO     .EQ $6900
9400:          00160 NDLIST   .EQ $9400 ;ADR OF NEW DISPLAY LIST
9000:          00165 CHRSET   .EQ $9000 ;ADR OF CHARACTER SET
               00170 *PLAYER MISSILE EQUATES
D407:          00175 PMBASE   .EQ $D407
8800:          00180 PDATA    .EQ $8800
D01D:          00185 GRACTL   .EQ $D01D
022F:          00190 DMACTL   .EQ $22F
D00C:          00195 SIZEM    .EQ $D00C
D008:          00200 SIZEP0   .EQ $D008
02C0:          00205 PCOLR0   .EQ $2C0
02C1:          00210 PCOLR1   .EQ $2C1
02C2:          00215 PCOLR2   .EQ $2C2
02C3:          00220 PCOLR3   .EQ $2C3
D000:          00225 HPOSP0   .EQ $D000
D001:          00230 HPOSP1   .EQ $D001
D002:          00235 HPOSP2   .EQ $D002
D003:          00240 HPOSP3   .EQ $D003
D004:          00245 HPOSM0   .EQ $D004
D005:          00250 HPOSM1   .EQ $D005
D006:          00255 HPOSM2   .EQ $D006
D007:          00260 HPOSM3   .EQ $D007
               00265 *COLLISIONS
D000:          00270 M0PF     .EQ $D000
D004:          00275 P0PF     .EQ $D004
D005:          00280 P1PF     .EQ $D005
D006:          00285 P2PF     .EQ $D006
D007:          00290 P3PF     .EQ $D007
D008:          00295 M0PL     .EQ $D008
D009:          00300 M1PL     .EQ $D009
D00A:          00305 M2PL     .EQ $D00A
D00C:          00310 P0PL     .EQ $D00C
D01E:          00315 HITCLR   .EQ $D01E
               00320 *MISC EQUATES
E45C:          00325 SETVBK   .EQ $E45C
E462:          00330 XITVBK   .EQ $E462
0278:          00335 STICK    .EQ $278
0284:          00340 STRIG0   .EQ $284
D01F:          00345 CONSOL   .EQ $D01F
D409:          00350 CHBASE   .EQ $D409
D404:          00355 HSCROL   .EQ $D404
D40A:          00360 WSYNC    .EQ $D40A
D20A:          00365 RANDOM   .EQ $D20A
02C4:          00370 COLOR0   .EQ $2C4
02C5:          00375 COLOR1   .EQ $2C5
02C8:          00380 COLOR4   .EQ $2C8
D200:          00385 AUDF1    .EQ $D200 ;USE FOR LASER
D201:          00390 AUDC1    .EQ $D201
D202:          00395 AUDF2    .EQ $D202 ;USE FOR EXPLOSIONS ALIENS & TARGETS
D203:          00400 AUDC2    .EQ $D203
D204:          00405 AUDF3    .EQ $D204 ;USE FOR SHIP ENGINE
```

```
D205:           00410 AUDC3    .EQ $D205
D206:           00415 AUDF4    .EQ $D206       ;USE FOR SHIP EXPLOSION
D207:           00420 AUDC4    .EQ $D207
D208:           00425 AUDCTL   .EQ $D208
D20F:           00430 SKCTL    .EQ $D20F
                00435 *SCREEN ENCODED DATA
4000: 00 00     00440 VALUE    .HS 0000              ;ROW 0
4002: 00 00     00445          .HS 0000              ;ROW 1
4004: 00 00     00450          .HS 0000              ;ROW 2
4006: 00 00     00455          .HS 0000              ;ROW 3
4008: 00 00     00460          .HS 0000              ;ROW 4
400A: 00 00     00465          .HS 0000              ;ROW 5
400C: 00 00     00470          .HS 0000              ;ROW 6
400E: 00 00     00475          .HS 0000              ;ROW 7
4010: 00 00     00480          .HS 0000              ;ROW 8
4012: 00 00     00485          .HS 0000              ;ROW 9
4014: 00 00     00490          .HS 0000              ;ROW 10
4016: 00 00     00495          .HS 0000              ;ROW 11
4018: 00 00     00500          .HS 0000              ;ROW 12
401A: 00 03 04
401D: 01 07 08
4020: 00        00505          .HS 00030401070800    ;ROW 13
4021: 00 04 01
4024: 09 00     00510          .HS 0004010900        ;ROW 14
4026: 00 03 02
4029: 01 0A 00
402C: 04 01     00515          .HS 000302010A000401 ;ROW 15
402E: 09 00     00520          .HS 0900
4030: 00 04 07
4033: 08 00 03
4036: 02 0A     00525          .HS 000407080003020A ;ROW 16
4038: 00 03 02
403B: 01 0B 00
403E: 04 01     00530          .HS 000302010B000401
4040: 09 00 04
4043: 0A 05 0A
4046: 00        00535          .HS 0900040A050A00
4047: 00 04 01
404A: 09 00 05
404D: 01 0B     00540          .HS 000401090005010B ;ROW 17
404F: 00 04 01
4052: 0A 00 04
4055: 09 00     00545          .HS 0004010A00040900
4057: 05 01 09
405A: 00 03 02
405D: 0A 00     00550          .HS 0501090003020A00
405F: 04 01 0B
4062: 06 0B 00 00555           .HS 04010B060B00
4065: 00 05 01
4068: 09 00 06
406B: 01 07     00560          .HS 0005010900060107 ;ROW 18
406D: 08 00 04
4070: 01 0B 00
4073: 03 02     00565          .HS 080004010B000302
4075: 01 09 00
4078: 06 01 0A
407B: 00 05     00570          .HS 01090006010A0005
407D: 01 0B 04
4080: 01 0A 00 00575           .HS 010B04010A00
4083: 00 06 01
4086: 09 00 04
```

```
4089: 01 09     00580           .HS 0006010900040109 ;ROW 19
408B: 00 01 07
408E: 08 00 03
4091: 02 01     00585           .HS 0001070800030201
4093: 09 00 04
4096: 01 0B 00
4099: 06 01     00590           .HS 090004010B000601
409B: 0B 00     00595           .HS 0B00
409D: 00 03 02
40A0: 09 00 03
40A3: 02 0A     00600           .HS 000302090003020A ;ROW 20
40A5: 00 04 01
40A8: 07 08 00
40AB: 03 02     00605           .HS 0004010708000302
40AD: 01 09 00
40B0: 03 02 07
40B3: 08 00     00610           .HS 0109000302070800
40B5: 0C 01 09
40B8: 00 04 01
40BB: 0A 00     00615           .HS 0C01090004010A00
40BD: 03 02 01
40C0: 0A 00 05
40C3: 01 07     00620           .HS 0302010A00050107
40C5: 08 00 03
40C8: 02 09 00 00625            .HS 080003020900
40CB: 00 04 01
40CE: 09 00 04
40D1: 01 0B     00630           .HS 000401090004010B ;ROW 21
40D3: 00 05 01
40D6: 09 00 04
40D9: 01 00     00635           .HS 0005010900040100
40DB: 0C 01 09
40DE: 00 04 01
40E1: 0B 00     00640           .HS 0C01090004010B00
40E3: 05 01 0B
40E6: 00 06 01
40E9: 0A 00     00645           .HS 05010B0006010A00
40EB: 04 09 04
40EE: 09 00 04
40F1: 01 09     00650           .HS 0409040900040109
40F3: 00 03 02
40F6: 01 07 08
40F9: 00 04     00655           .HS 0003020107080004 ;ROW 22
40FB: 01 07 08
40FE: 00 04 01
4101: 00 04     00660           .HS 0107080004010004
4103: 01 09 00
4106: 04 01 07
4109: 08 00     00665           .HS 0109000401070800
410B: 06 01 0B
410E: 04 01 09
4111: 00 03     00670           .HS 06010B0401090003
4113: 02 01     00675           .HS 0201
4115: 01 01     00680           .HS 0101            ;ROW 23
4117:           00685           .BS $E9
4200: FF 01     00690 BLOCKS    .HS FF01            ;ROW 0
4202: FF 01     00695           .HS FF01            ;ROW 1
4204: FF 01     00700           .HS FF01            ;ROW 2
4206: FF 01     00705           .HS FF01            ;ROW 3
4208: FF 01     00710           .HS FF01            ;ROW 4
420A: FF 01     00715           .HS FF01            ;ROW 5
```

```
420C: FF 01     00720           .HS FF01              ;ROW 6
420E: FF 01     00725           .HS FF01              ;ROW 7
4210: FF 01     00730           .HS FF01              ;ROW 8
4212: FF 01     00735           .HS FF01              ;ROW 9
4214: FF 01     00740           .HS FF01              ;ROW 10
4216: FF 01     00745           .HS FF01              ;ROW 11
4218: FF 01     00750           .HS FF01              ;ROW 12
421A: B7 01 01
421D: 05 01 01
4220: 40        00755           .HS B7010105010140    ;ROW 13
4221: B6 01 09
4224: 01 3F     00760           .HS B60109013F        ;ROW 14
4226: 77 01 01
4229: 07 01 34
422C: 01 0B     00765           .HS 7701010701340108 ;ROW 15
422E: 01 3E     00770           .HS 013E
4230: 30 01 01
4233: 01 0F 01
4236: 01 01     00775           .HS 300101010F010101 ;ROW 16
4238: 2E 01 01
423B: 0B 01 33
423E: 01 0D     00780           .HS 2E01010B0133010D
4240: 01 19 01
4243: 01 01 01
4246: 20        00785           .HS 01190101010120
4247: 2F 01 03
424A: 01 0D 01
424D: 02 01     00790           .HS 2F0103010D010201 ;ROW 17
424F: 2D 01 0E
4252: 01 0E 01
4255: 01 21     00795           .HS 2D010E010E010121
4257: 01 0F 01
425A: 13 01 01
425D: 01 01     00800           .HS 010F011301010101
425F: 01 01 01
4262: 01 01 20 00805            .HS 010101010120
4265: 2E 01 05
4268: 01 0C 01
426B: 03 01     00810           .HS 2E0105010C010301 ;ROW 18
426D: 01 29 01
4270: 10 01 0C
4273: 01 01     00815           .HS 012901100108C0101
4275: 02 01 20
4278: 01 10 01
427B: 11 01     00820           .HS 0201200110011101
427D: 02 01 01
4280: 05 01 1F 00825            .HS 02010105011F
4283: 2E 01 06
4286: 01 0A 01
4289: 06 01     00830           .HS 2E0106010A010601 ;ROW 19
428B: 28 12 01
428E: 01 08 01
4291: 01 05     00835           .HS 2812010108010105
4293: 01 17 01
4296: 18 01 11
4299: 01 09     00840           .HS 0117011801110109
429B: 01 1F     00845           .HS 011F
429D: 10 01 01
42A0: 01 0C 01
42A3: 01 01     00850           .HS 100101010C010101 ;ROW 20
42A5: 0B 01 08
```

```
42A8: 01 01 04
42AB: 01 01    00855          .HS 0B01080101040101
42AD: 0A 01 1B
42B0: 01 01 01
42B3: 01 08    00860          .HS 0A011B0101010108
42B5: 01 23 01
42B8: 15 01 1A
42BB: 01 08    00865          .HS 012301150110A0108
42BD: 01 01 02
42C0: 01 02 01
42C3: 0B 01    00870          .HS 0101020102010B01
42C5: 01 19 01
42C8: 01 01 01 00875          .HS 011901010101
42CB: 0F 01 03
42CE: 01 0A 01
42D1: 02 01    00880          .HS 0F0103010A010201 ;ROW 21
42D3: 0A 01 1C
42D6: 01 19 01
42D9: 04 0B    00885          .HS 0A011C011901040B
42DB: 01 21 01
42DE: 0C 01 22
42E1: 01 07    00890          .HS 0121010C01220107
42E3: 01 04 01
42E6: 02 01 0D
42E9: 01 01    00895          .HS 01040102010D0101
42EB: 01 01 01
42EE: 01 12 01
42F1: 03 01    00900          .HS 0101010112010301
42F3: 0D 01 01
42F6: 05 01 01
42F9: 07 01    00905          .HS 0D01010501010701 ;ROW 22
42FB: 3A 01 01
42FE: 08 01 05
4301: 0B 01    00910          .HS 3A01010801050B01
4303: 28 01 04
4306: 01 24 01
4309: 01 05    00915          .HS 2801040124010105
430B: 01 15 01
430E: 01 04 01
4311: 0F 01    00920          .HS 0115010104010F01
4313: 01 05    00925          .HS 0105
4315: FF 01    00930          .HS FF01             ;ROW 23
4317:          00935          .BS $E9
               00940 *CHARACTER SET 512 BYTES LONG
4400: 00 00 00
4403: 00 00 00
4406: 00 00    00945 SETCHAR  .HS 0000000000000000
4408: FF FF FF
440B: FF FF FF
440E: FF FF    00950          .HS FFFFFFFFFFFFFFFF
4410: 03 0F 3F
4413: FF FF FF
4416: FF FF    00955          .HS 030F3FFFFFFFFFFF
4418: 00 00 00
441B: 00 03 0F
441E: 3F FF    00960          .HS 00000000030F3FFF
4420: 01 03 07
4423: 0F 1F 3F
4426: 7F FF    00965          .HS 0103070F1F3F7FFF
4428: 01 01 03
442B: 03 07 07
```

```
442E: 0F 0F     00970           .HS 0101030307070F0F ;#5
4430: 1F 1F 3F
4433: 3F 7F 7F
4436: FF FF     00975           .HS 1F1F3F3F7F7FFFFF
4438: C0 F0 FC
443B: FF FF FF
443E: FF FF     00980           .HS C0F0FCFFFFFFFFFF
4440: 00 00 00
4443: 00 C0 F0
4446: FC FF     00985           .HS 00000000C0F0FCFF
4448: 80 C0 E0
444B: F0 F8 FC
444E: FE FF     00990           .HS 80C0E0F0F8FCFEFF
4450: 80 80 C0
4453: C0 E0 E0
4456: F0 F0     00995           .HS 8080C0C0E0E0F0F0 ;#10
4458: F8 F8 FC
445B: FC FE FE
445E: FF FF     01000           .HS F8F8FCFCFEFEFFFF
4460: FF 7F 3F
4463: 1F 0F 07
4466: 03 01     01005           .HS FF7F3F1F0F070301
4468:           01010           .BS $98
4500: 80 C1 67
4503: 3F 1F 1F
4506: 3F 3F     01015           .HS 80C1673F1F1F3F3F ;#32
4508: 00 C0 F0
450B: F8 F8 FC
450E: FC FE     01020           .HS 00C0F0F8F8FCFCFE
4510: 80 40 20
4513: 10 08 04
4516: 02 01     01025           .HS 8040201008040201
4518: 3C 3C 3C
451B: 3C 7E 7E
451E: FF C3     01030           .HS 3C3C3C3C7E7EFFC3
4520: 18 18 18
4523: 18 18 18
4526: 3C 3C     01035           .HS 1818181818183C3C
4528: 00 3E 60
452B: 60 6E 66
452E: 3E 00     01040           .HS 003E60606E663E00 ;G
4530: 00 18 3C
4533: 66 66 7E
4536: 66 00     01045           .HS 00183C66667E6600 ;A
4538: 00 63 77
453B: 7F 6B 63
453E: 63 00     01050           .HS 0063777F6B636300 ;M
4540: 00 7E 60
4543: 7C 60 60
4546: 7E 00     01055           .HS 007E607C60607E00 ;E
4548: 00 3C 66
454B: 66 66 66
454E: 3C 00     01060           .HS 003C666666663C00 ;O
4550: 00 66 66
4553: 66 66 3C
4556: 18 00     01065           .HS 00666666663C1800 ;V
4558: 00 7C 66
455B: 66 7C 6C
455E: 66 00     01070           .HS 007C66667C6C6600 ;R
4560:           01075           .BS $A0
                01080 *PLAYER#1 PROGRAMABLE BLOCK
```

```
4600: D0 50 30 01085 ENEMY1   .HS D05030          ;X,Y,DELAY
4603: 28 00 FF
4606: 00 00 00 01090          .HS 2800FF000000    ;SHAPE#0
4609: 50 00 FF
460C: 01 00 00 01095          .HS 5000FF010000
460F: 70 00 FF
4612: 00 01 07 01100          .HS 7000FF000107
4615: FF 00 FF
4618: 00 00 00 01105          .HS FF00FF000000
461B: 00 00 00
461E: 00 00 00 01110          .HS 000000000000
4621: 00 00 00
4624: 00 00 00 01115          .HS 000000000000
4627: 00 00 00
462A: 00 00 00 01120          .HS 000000000000
462D: 00 00 00
4630: 00 00 00 01125          .HS 000000000000
4633: D0 75 40 01130          .HS D07540          ;SHAPE#1
4636: 30 00 FF
4639: FF 00 00 01135          .HS 3000FFFF0000
463C: 55 00 00
463F: 01 00 00 01140          .HS 550000010000
4642: 65 00 FF
4645: FF 00 00 01145          .HS 6500FFFF0000
4648: 75 00 FF
464B: FF 01 06 01150          .HS 7500FFFF0106
464E: 80 00 FF
4651: 01 01 05 01155          .HS 8000FF010105
4654: 98 00 FF
4657: FF 00 00 01160          .HS 9800FFFF0000
465A: FF 00 00
465D: FF 01 03 01165          .HS FF0000FF0103
4660: 00 00 00
4663: 00 00 00 01170          .HS 000000000000
4666: 50 32 25 01175          .HS 503225          ;SHAPE#2
4669: 15 00 01
466C: 01 00 00 01180          .HS 150001010000
466F: 30 00 01
4672: 01 01 05 01185          .HS 300001010105
4675: 40 00 01
4678: 00 01 06 01190          .HS 400001000106
467B: 53 00 01
467E: 00 00 00 01195          .HS 530001000000
4681: 70 00 01
4684: FF 00 00 01200          .HS 700001FF0000
4687: 80 00 FF
468A: 00 00 00 01205          .HS 8000FF000000
468D: A8 00 FF
4690: 01 01 05 01210          .HS A800FF010105
4693: FF 01 FF
4696: 00 01 03 01215          .HS FF01FF000103
4699: D0 70 35 01220          .HS D07035          ;SHAPE#3
469C: 20 00 FF
469F: FF 00 00 01225          .HS 2000FFFF0000
46A2: 40 00 FF
46A5: 01 00 00 01230          .HS 4000FF010000
46A8: 60 00 FF
46AB: FF 01 06 01235          .HS 6000FFFF0106
46AE: 80 00 FF
46B1: FF 01 06 01240          .HS 8000FFFF0106
46B4: 98 00 FF
```

```
46B7: 01 01 07 01245          .HS 9800FF010107
46BA: FF 00 FF
46BD: FF 01 01 01250          .HS FF00FFFF0101
46C0: 00 00 00
46C3: 00 00 00 01255          .HS 000000000000
46C6: 00 00 00
46C9: 00 00 00 01260          .HS 000000000000
46CC: D0 65 15 01265          .HS D06515          ;SHAPE#4
46CF: 25 00 FF
46D2: 00 00 00 01270          .HS 2500FF000000
46D5: 40 00 FF
46D8: 01 00 00 01275          .HS 4000FF010000
46DB: 60 00 FF
46DE: FF 01 06 01280          .HS 6000FFFF0106
46E1: 80 00 00
46E4: FF 01 06 01285          .HS 800000FF0106
46E7: A0 00 FF
46EA: 01 01 05 01290          .HS A000FF010105
46ED: FF 00 FF
46F0: 00 01 01 01295          .HS FF00FF000101
46F3: 00 00 00
46F6: 00 00 00 01300          .HS 000000000000
46F9: 00 00 00
46FC: 00 00 00
46FF: 00       01305          .HS 00000000000000
4700: D0 40 50 01310 ENEMY2   .HS D04050          ;X,Y,DELAY
4703: 38 00 FF
4706: 01 00 00 01315          .HS 3800FF010000    ;SHAPE#0
4709: 50 00 FE
470C: FF 01 06 01320          .HS 5000FEFF0106
470F: 68 00 FF
4712: FF 00 00 01325          .HS 6800FFFF0000
4715: 78 00 FF
4718: 01 00 00 01330          .HS 7800FF010000
471B: 90 00 FF
471E: 00 01 03 01335          .HS 9000FF000103
4721: FF 00 FF
4724: 00 00 00 01340          .HS FF00FF000000
4727: 00 00 00
472A: 00 00 00 01345          .HS 000000000000
472D: 00 00 00
4730: 00 00 00 01350          .HS 000000000000
4733: D0 35 20 01355          .HS D03520          ;SHAPE#1
4736: 18 00 FF
4739: 01 00 00 01360          .HS 1800FF010000
473C: 30 00 FF
473F: FF 00 00 01365          .HS 3000FFFF0000
4742: 40 00 FF
4745: 00 00 00 01370          .HS 4000FF000000
4748: 60 00 FF
474B: 01 01 06 01375          .HS 6000FF010106
474E: 70 00 01
4751: 00 01 07 01380          .HS 700001000107
4754: 80 00 FF
4757: FF 01 07 01385          .HS 8000FFFF0107
475A: FF 00 FF
475D: 01 01 03 01390          .HS FF00FF010103
4760: 00 00 00
4763: 00 00 00 01395          .HS 000000000000
4766: D0 80 20 01400          .HS D08020          ;SHAPE#2
4769: 30 00 FF
```

```
476C: 00 00 00 01405          .HS 3000FF000000
476F: 50 00 FF
4772: FF 00 00 01410          .HS 5000FFFF0000
4775: 60 00 FF
4778: FF 01 07 01415          .HS 6000FFFF0107
477B: 78 00 FF
477E: 01 01 05 01420          .HS 7800FF010105
4781: FF 00 FF
4784: FF 01 03 01425          .HS FF00FFFF0103
4787: 00 00 00
478A: 00 00 00 01430          .HS 000000000000
478D: 00 00 00
4790: 00 00 00 01435          .HS 000000000000
4793: 00 00 00
4796: 00 00 00 01440          .HS 000000000000
4799: D0 55 20 01445          .HS D05520          ;SHAPE#3
479C: 15 00 FF
479F: 01 00 00 01450          .HS 1500FF010000
47A2: 35 00 FE
47A5: 00 00 00 01455          .HS 3500FE000000
47A8: 45 00 01
47AB: FF 00 00 01460          .HS 450001FF0000
47AE: 55 00 00
47B1: FF 01 06 01465          .HS 550000FF0106
47B4: 70 00 FF
47B7: 00 00 00 01470          .HS 7000FF000000
47BA: 95 00 FF
47BD: 00 01 05 01475          .HS 9500FF000105
47C0: FF 00 FF
47C3: 00 01 03 01480          .HS FF00FF000103
47C6: 00 00 00
47C9: 00 00 00 01485          .HS 000000000000
47CC: D0 65 20 01490          .HS D06520          ;SHAPE#4
47CF: 20 00 FF
47D2: 00 00 00 01495          .HS 2000FF000000
47D5: 38 00 FF
47D8: FF 00 00 01500          .HS 3800FFFF0000
47DB: 50 00 FF
47DE: 01 01 06 01505          .HS 5000FF010106
47E1: 68 00 FF
47E4: 00 01 05 01510          .HS 6800FF000105
47E7: 78 00 FF
47EA: FF 01 03 01515          .HS 7800FFFF0103
47ED: FF 00 FF
47F0: 01 00 00 01520          .HS FF00FF010000
47F3: 00 00 00
47F6: 00 00 00 01525          .HS 000000000000
47F9: 00 00 00
47FC: 00 00 00
47FF: 00      01530           .HS 00000000000000
4800: 70 70 70
4803: 46 00 69
4806: 86 56   01535 DLIST     .HS 7070704600698656
4808: 00 72 56
480B: 00 73 56
480E: 00 74   01540           .HS 0072560073560074
4810: 56 00 75
4813: 56 00 76
4816: 56 00   01545           .HS 5600755600765600
4818: 77 56 00
481B: 78 56 00
```

```
481E: 79 56     01550           .HS 7756007856007956
4820: 00 7A 56
4823: 00 7B 56
4826: 00 7C     01555           .HS 007A56007B56007C
4828: 56 00 7D
482B: 56 00 7E
482E: 56 00     01560           .HS 56007D56007E5600
4830: 7F 56 00
4833: 80 56 00
4836: 81 56     01565           .HS 7F56008056008156
4838: 00 82 56
483B: 00 83 56
483E: 00 84     01570           .HS 0082560083560084
4840: 56 00 85
4843: 56 00 86
4846: 56 00     01575           .HS 5600855600865600
4848: 87 41 00
484B: 94        01580           .HS 87410094
484C: 80 80 C0
484F: FC F6 7F
4852: 7E 00     01585 SHIP      .HS 8080C0FCF67F7E00
4854: 3C 3C 3C
4857: 7E DB DB
485A: DB DB     01590 ALIEN     .HS 3C3C3C7EDBDBDBDB
485C: 81 42 3C
485F: 3C 3C 3C
4862: 42 81     01595           .HS 81423C3C3C3C4281
4864: 3C 3C 18
4867: 18 99 FF
486A: C3 81     01600           .HS 3C3C181899FFC381
486C: 91 7E 46
486F: C2 43 62
4872: 7E 89     01605           .HS 917E46C243627E89
4874: 00 00 3C
4877: 7E FF 7E
487A: 3C 00     01610           .HS 00003C7EFF7E3C00
487C: 54 5C 64
487F: 6C 74     01615 ALIENPT   .HS 545C646C74
4881: 00 33 66
4884: 99 CC     01620 E1PT      .HS 00336699CC
4886: 00 33 66
4889: 99 CC     01625 E2PT      .HS 00336699CC
488B: 03 00 0C
488E: 0C 30 30
4891: C0 C0     01630 MSHAPE    .HS 03000C0C3030C0C0 ;4 MISSILES EACH TWO HIGH
4893: 8B 8D 8F
4896: 91        01635 MISLO     .HS 8B8D8F91
4897: F8 3E F8
489A: 00 00 00
489D: 00 00     01640 BOMBSH    .HS F83EF80000000000
489F: 00 01 01
48A2: 01 00 FF
48A5: FF FF     01645 VMX       .HS 0001010100FFFFFF
48A7: FF FF 00
48AA: 01 01 01
48AD: 00 FF     01650 VMY       .HS FFFF0001010100FF
48AF: 2B 4A 69
48B2: 84 97 A9
48B5: D3        01655 MBPOSL    .HS 2B4A698497A9D3
48B6: 84 84 85
48B9: 82 84 83
```

```
48BC: 84       01660 MBPOSH   .HS 84848582848384
48BD: 00 00 00
48C0: 00 33 34
48C3: 32 29 2B
48C6: 25 00 26
48C9: 2F 32 23
48CC: 25 00 00
48CF: 00 00    01665 TITLE    .AT '    STRIKE FORCE    '
48D1: 33 28 29
48D4: 30 33 00
48D7: 10 00 00
48DA: 33 23 2F
48DD: 32 25 00
48E0: 10 10 10
48E3: 10 10    01670          .AT 'SHIPS 0  SCORE 00000'
48E5: 20 42 59
48E8: 20 4A 45
48EB: 46 46 52
48EE: 45       01675 TITLE1   .HS 204259204A4546465245
48EF: 59 20 53
48F2: 54 41 4E
48F5: 54 4F 4E
48F8: 20       01680          .HS 592053544414E544F4E20
48F9:          01685          .BS $07
4900: 65 66 67
4903: 68 00 69
4906: 6A 68 6B 01690 GOVER    .HS 6566676800696A686B ;GAME OVER
               01695 *VARIABLES
4909:          01700 XPMO     .BS 1     ;ACTUAL X POS ON SCREEN
490A:          01705 YPMO     .BS 1     ;ACTUAL Y POS ON SCREEN
490B:          01710 XPL      .BS 1     ;SHIP POS IN WORLD 0-1060
490C:          01715 XPH      .BS 1
490D:          01720 XS       .BS 1     ;BACKGROUND AT LEFT EDGE
490E: 00       01725 FS       .DA #0    ;FINE SCROLL REG
490F:          01730 YMISOLD0 .BS 1     ;OLD Y VALUE FOR MISSILE #0
4910:          01735 YMISOLD1 .BS 1
4911:          01740 YMISOLD2 .BS 1
4912:          01745 INDEX1   .BS 1     ;USED FOR TEMP STORAGE
4913:          01750 INDEX2   .BS 1
4914: 00       01755 COUNT    .DA #0    ;COUNTER DURING DATA UNPACK
4915:          01760 TEMP     .BS 1     ;TEMP STORAGE
4916: 00       01765 BACK     .DA #0    ;STICK BACK FLAG
4917:          01770 SPEED    .BS 1     ;SHIP SPEED FLAG - FAST OR SLOW
4918:          01775 BOMBON   .BS 1     ;FLAG SET WHEN BOMB DROPPING
4919: 00 00    01780 VBFLAG   .DA 0     ;FLAG ON UPON ENTERING VBLANK
491B:          01785 XSP      .BS 1     ;BACKGROUND AT SHIP
491C: 00       01790 ONSCRN1  .DA #0    ;ALIEN SHIP ON SCREEN FLAG
491D: 00       01795 ONSCRN2  .DA #0
491E: 28       01800 TDELAY1  .DA #40   ;CURRENT DELAY BEFORE ALIEN APPEARS
491F:          01805 TDELAY2  .BS 1
4920:          01810 NDELAY1  .BS 1     ;DELAY FOR NEXT ALIEN TO APPEAR
4921:          01815 NDELAY2  .BS 1
4922:          01820 TIMER1L  .BS 1     ;ALIEN SHIP #1 TIMER
4923:          01825 TIMER1H  .BS 1
4924:          01830 TIMER2L  .BS 1     ;ALIEN SHIP #2 TIMER
4925:          01835 TIMER2H  .BS 1
4926:          01840 TIME1L   .BS 1     ;WHEN TO READ NEXT ALIEN#1 INSTRUCTION
4927:          01845 TIME2L   .BS 1
4928:          01850 TIME1H   .BS 1
4929:          01855 TIME2H   .BS 1
492A:          01860 TMISO    .BS 1
```

```
492B:          01865 TMIS1    .BS 1     ;ALIEN #1 MISSILE TIMER
492C:          01870 TMIS2    .BS 1
492D:          01875 VX0      .BS 1
492E:          01880 VX1      .BS 1     ;VELOCITY ALIEN #1
492F:          01885 VX2      .BS 1
4930:          01890 VX3      .BS 1     ;VELOCITY BOMB
4931:          01895 VY0      .BS 1
4932:          01900 VY1      .BS 1     ;VELOCITY ALIEN #1
4933:          01905 VY2      .BS 1
4934:          01910 VY3      .BS 1     ;VELOCITY BOMB
4935:          01915 VTEMP    .BS 1
4936:          01920 ACCEL    .BS 1     ;BOMB ACCELERATION
4937:          01925 X0       .BS 1
4938:          01930 X1       .BS 1     ;ALIEN #1 POSITION
4939:          01935 X2       .BS 1
493A:          01940 XB       .BS 1     ;HORIZ POSITION BOMB
493B:          01945 Y0       .BS 1
493C:          01950 Y1       .BS 1     ;ALIEN #1 POSITION
493D:          01955 Y2       .BS 1
493E:          01960 YB       .BS 1     ;VERT POSITION BOMB
493F:          01965 X0M      .BS 1     ;HORIZ MISSILE #0 POSITION
4940:          01970 X1M      .BS 1
4941:          01975 X2M      .BS 1
4942:          01980 Y0M      .BS 1     ;VERT MISSILE #0 POSITION
4943:          01985 Y1M      .BS 1
4944:          01990 Y2M      .BS 1
4945:          01995 SHOOT0   .BS 1
4946:          02000 SHOOT1   .BS 1     ;FLAG FOR ALIEN #1 TO SHOOT
4947:          02005 SHOOT2   .BS 1
4948:          02010 TEMPL    .BS 4     ;TEMP STORAGE DURING PLOTTING
494C:          02015 TEMPH    .BS 4
4950:          02020 DIR0     .BS 1
4951:          02025 DIR1     .BS 1     ;DIRECTION TO SHOOT ALIEN #1
4952:          02030 DIR2     .BS 1
4953:          02035 MDIR0    .BS 1
4954:          02040 MDIR1    .BS 1     ;TEMP STORAGE DURING MISSILE TRACK
4955:          02045 MDIR2    .BS 1
4956:          02050 MASK     .BS 1     ;MISSILE #0 MASK
4957: FC       02055 MASK0    .DA #$FC  ;MISSILE #0 MASK
4958: F3       02060 MASK1    .DA #$F3
4959: CF       02065 MASK2    .DA #$CF
495A:          02070 LASON    .BS 4     ;LASER ON FLAG
495E:          02075 LPOSL    .BS 4     ;ACTUAL SCREEN POSITION OF LASER -LO
4962:          02080 LPOSH    .BS 4
4966:          02085 LDELAY   .BS 4     ;DELAY UNTIL LASER BASE REFIRES
496A:          02090 LCOUNT   .BS 4     ;DELAY BETWEEN MOVEMENT OF LASER BEAM
496E:          02095 GALIVE   .BS 4     ;LASER BASE ON FLAG
4972:          02100 YG       .BS 4     ;LASER BASE Y POSITION
4976:          02105 XG       .BS 4     ;LASER BASE X POSITION
497A:          02110 SR10     .BS 1     ;SCORE 10'S DIGIT
497B:          02115 SR100    .BS 1
497C:          02120 SR1000   .BS 1
497D:          02125 SR10T    .BS 1
497E:          02130 HSR10    .BS 1     ;HIGH SCORE 10'S DIGIT
497F:          02135 HSR100   .BS 1
4980:          02140 HSR1000  .BS 1
4981:          02145 HSR10T   .BS 1
4982:          02150 DELAY    .BS 1     ;DELAY FLAG ON WHEN SHIP EXPLODES
4983:          02155 SHIPNUM  .BS 1     ;NUMBER OF SHIPS
4984:          02160 BTARGET  .BS 1     ;PLAYFIELD POSITION OF BOMB
4985:          02165 STRIKE   .BS 1     ;COUNTER FOR TARGET HIT
```

```
4986:           02170 MBCOUNT  .BS 1  ;# MISSILE BASES HIT
4987:           02175 LUCK     .BS 1  ;RANDOM # 0-4
4988:           02180 RDSHP1   .BS 1  ;RANDOM SHAPE # ALIEN #1
4989:           02185 RDSHP2   .BS 1
498A:           02190 POS      .BS 1  ;HORIZ BOMB POSITION
498B:           02195 T        .BS 1  ;FLAG FOR WHICH PART OF TITLE IS ON SCREEN
498C:           02200 OVER     .BS 1  ;OFFSET TO WHERE GAME OVER TITLE IS WRITTEN
498D: 04        02205 DELAY1   .HS 04
498E:           02210 SLTIME   .BS 1  ;COUNTDOWN TIMER FOR LASER SOUND
498F:           02215 SLTIME1  .BS 1
4990:           02220 SETIME   .BS 1  ;COUNTDOWN TIMER FOR EXPLOSION
4991:           02225 SEXTIME  .BS 1  ;COUNTDOWN TIMER FOR SHIP EXPLOSION
4992:           02230 REZFLAG  .BS 1  ;DEREZ ON FLAG
4993:           02235 EXCOUNT  .BS 1  ;COUNTER DURING DEREZ
4994:           02240 MEDIUM   .BS 1  ;PLAY MEDIUM DIFFICULTY FLAG
4995:           02245 HARD     .BS 1  ;PLAY HARD DIFFICULTY
4996:           02250 DEREZ    .BS 8  ;DUMMY AREA FOR SHUP DURING DEREZ
                02255          .IN "D:SCROLL1B"
                02500 *PART 2 OF SCROLLING GAME
                02505 *ZERO HIGH SCORE & SCORE
499E: A9 10     02510          LDA #$10 ;SCORES ARE OFFSET BY #$10 TO MATCH CHARACTER SET
49A0: 8D 7A 49  02515          STA SR10
49A3: 8D 7B 49  02520          STA SR100
49A6: 8D 7C 49  02525          STA SR1000
49A9: 8D 7D 49  02530          STA SR10T
49AC: 8D 7E 49  02535          STA HSR10
49AF: 8D 7F 49  02540          STA HSR100
49B2: 8D 80 49  02545          STA HSR1000
49B5: 8D 81 49  02550          STA HSR10T
                02555 *SET SYSTEM RESET VECTOR
49B8: A9 49     02560          LDA /START
49BA: 85 03     02565          STA $03
49BC: A9 C7     02570          LDA #START
49BE: 85 02     02575          STA $02
49C0: A9 02     02580          LDA #$02    ;TELLS SYS RESET A CASSETTE
49C2: 85 09     02585          STA $09
49C4: 4C 74 E4  02590          JMP $E474   ;SYSTEM RESET VECTOR
                02595 *PART 2 OF SCROLLING GAME
                02600 *TERRAIN GENERATOR ROUTINE
49C7: A9 70     02605 START    LDA #$70    ;SCREEN AT $7000
49C9: 85 F7     02610          STA GROUNDH
49CB: A9 00     02615          LDA #$00
49CD: 85 F6     02620          STA GROUNDL
49CF: A9 40     02625          LDA /VALUE
49D1: 85 F9     02630          STA VALUEH
49D3: A9 00     02635          LDA #VALUE
49D5: 85 F8     02640          STA VALUEL
49D7: A9 42     02645          LDA /BLOCKS
49D9: 85 FB     02650          STA BLOCKH
49DB: A9 00     02655          LDA #BLOCKS
49DD: 85 FA     02660          STA BLOCKL
49DF: A9 00     02665          LDA #$00
49E1: 8D 14 49  02670          STA COUNT
49E4: A0 00     02675          LDY #$00
49E6: B1 FA     02680 LOOP     LDA (BLOCKL),Y ;#BLOCKS IN AROW
49E8: AA        02685          TAX         ;USE AS COUNTER
49E9: B1 F8     02690          LDA (VALUEL),Y ;LOAD BLOCK VALUE
49EB: 48        02695          PHA
49EC: 98        02700          TYA
49ED: 8D 15 49  02705          STA TEMP    ;SAVE Y REGISTER
49F0: AC 14 49  02710          LDY COUNT   ;BYTE COUNTER FOR ROW
```

```
49F3: 68       02715          PLA
49F4: 91 F6    02720 .1       STA (GROUNDL),Y ;STORE ON SCREEN MAP
49F6: C8       02725          INY         ;FOR NEXT LOCATION
49F7: CA       02730          DEX
49F8: D0 FA    02735          BNE .1      ;LOOP TILL DONE WITH # BLOCKS IN AROW
49FA: 98       02740          TYA
49FB: 8D 14 49 02745          STA COUNT
49FE: C0 00    02750          CPY #$00    ;END OF ROW?
4A00: D0 08    02755          BNE .2
4A02: E6 F7    02760          INC GROUNDH ;NEXT ROW
4A04: A5 F7    02765          LDA GROUNDH
4A06: C9 88    02770          CMP #$88    ;DONE WITH ALL 22 ROWS
4A08: B0 0F    02775          BGE DONE
4A0A: AC 15 49 02780 .2       LDY TEMP    ;RESTORE INDEX
4A0D: C8       02785          INY         ;UP INDEX FOR VALUES IN TABLES
4A0E: C0 00    02790          CPY #$00    ;CROSSED PAGE?
4A10: D0 D4    02795          BNE LOOP
4A12: E6 F9    02800          INC VALUEH  ;YES NEXT PAGE
4A14: E6 FB    02805          INC BLOCKH
4A16: 4C E6 49 02810          JMP LOOP
4A19: EA       02815 DONE     NOP
               02820 *SETUP DLIST
4A1A: A2 00    02825          LDX #$00
4A1C: BD 00 48 02830 DLOOP    LDA DLIST,X
4A1F: 9D 00 94 02835          STA NDLIST,X
4A22: E8       02840          INX
4A23: E0 4C    02845          CPX #$4C    ;76 ELEMENTS
4A25: D0 F5    02850          BNE DLOOP
4A27: A9 00    02855          LDA #NDLIST
4A29: 8D 30 02 02860          STA 560
4A2C: A9 94    02865          LDA /NDLIST
4A2E: 8D 31 02 02870          STA 561
               02875 *INITILIZE SHIP
4A31: A9 88    02880          LDA #$88
4A33: 8D 07 D4 02885          STA PMBASE
4A36: A9 19    02890          LDA #$19    ;9 SHIPS
4A38: 8D 83 49 02895          STA SHIPNUM
4A3B: A9 03    02900          LDA #$03    ;SET P/M GRAPHICS
4A3D: 8D 1D D0 02905          STA GRACTL
4A40: A9 3E    02910          LDA #$3E    ;ENABLE P/M DMA SINGLE LINE
4A42: 8D 2F 02 02915          STA DMACTL
4A45: A9 01    02920          LDA #$01    ;DOUBLE WIDTH
4A47: 8D 08 D0 02925          STA SIZEP0
4A4A: A9 7A    02930          LDA #$7A    ;PLAYER #0 122 BLUE-LUM 10
4A4C: 8D C0 02 02935          STA PCOLR0
4A4F: A9 00    02940          LDA #$00    ;INITIAL SCREEN LEFT
4A51: 8D 0D 49 02945          STA XS
4A54: A9 50    02950          LDA #$50    ;INITIAL POS SHIP X=80
4A56: 8D 09 49 02955          STA XPM0
4A59: 8D 00 D0 02960          STA HPOSP0  ; TELL ANTIC
4A5C: A9 60    02965          LDA #$60    ;INITIAL POS SHIP Y=96
4A5E: 8D 0A 49 02970          STA YPM0
4A61: A9 88    02975          LDA /PDATA
4A63: 18       02980          CLC
4A64: 69 04    02985          ADC #$04
4A66: 85 F5    02990          STA SHPMOH
               02995 *OTHER P/M CONSTANTS
4A68: A9 C6    03000          LDA #$C6    ;GREEN ALIEN
4A6A: 8D C1 02 03005          STA PCOLR1
4A6D: A9 44    03010          LDA #$44    ;RED ALIEN
4A6F: 8D C2 02 03015          STA PCOLR2
```

```
4A72: A9 7A    03020          LDA #$7A    ;BLUE BOMB
4A74: 8D C3 02 03025          STA PCOLR3
4A77: A9 E8    03030          LDA #$E8    ;INITIAL POSITIONS OF ALIENS
4A79: 8D 01 D0 03035          STA HPOSP1
4A7C: 8D 02 D0 03040          STA HPOSP2
4A7F: A9 03    03045          LDA #$03    ;MISSILE 0 QUADRUPLE-REST NORMAL
4A81: 8D 0C D0 03050          STA SIZEM
               03055 *CLEAR P/M AREA
4A84: A9 00    03060          LDA #$00    ;PDATAL
4A86: 85 FC    03065          STA PMADR
4A88: A9 88    03070          LDA /PDATA
4A8A: 85 FD    03075          STA PMADR+1
4A8C: A0 00    03080          LDY #$00
4A8E: 98       03085          TYA
4A8F: A2 08    03090          LDX #$08
4A91: 91 FC    03095 .1       STA (PMADR),Y
4A93: C8       03100          INY
4A94: D0 FB    03105          BNE .1
4A96: E6 FD    03110          INC PMADR+1 ;NEXT 256 BYTES
4A98: CA       03115          DEX
4A99: D0 F6    03120          BNE .1
               03125 *PUT CHARACTER SET IN POSITION
4A9B: A9 00    03130 MC       LDA #$00
4A9D: 85 FE    03135          STA CHADR
4A9F: 85 FC    03140          STA CHSET
4AA1: A9 90    03145          LDA /CHRSET
4AA3: 85 FF    03150          STA CHADR+1
4AA5: A9 44    03155          LDA /SETCHAR
4AA7: 85 FD    03160          STA CHSET+1
4AA9: A2 02    03165          LDX #$02
4AAB: A0 00    03170          LDY #$00
4AAD: B1 FC    03175 .1       LDA (CHSET),Y
4AAF: 91 FE    03180          STA (CHADR),Y
4AB1: C8       03185          INY
4AB2: D0 F9    03190          BNE .1
4AB4: E6 FD    03195          INC CHSET+1
4AB6: E6 FF    03200          INC CHADR+1
4AB8: CA       03205          DEX
4AB9: D0 F2    03210          BNE .1
               03215 *SETUP LASER BASES
4ABB: A9 3E    03220          LDA #$3E
4ABD: 8D 76 49 03225          STA XG
4AC0: A9 13    03230          LDA #$13
4AC2: 8D 72 49 03235          STA YG
4AC5: A9 71    03240          LDA #$71
4AC7: 8D 77 49 03245          STA XG+1
4ACA: A9 16    03250          LDA #$16
4ACC: 8D 73 49 03255          STA YG+1
4ACF: A9 9E    03260          LDA #$9E
4AD1: 8D 78 49 03265          STA XG+2
4AD4: A9 16    03270          LDA #$16
4AD6: 8D 74 49 03275          STA YG+2
4AD9: A9 CB    03280          LDA #$CB
4ADB: 8D 79 49 03285          STA XG+3
4ADE: A9 16    03290          LDA #$16
4AE0: 8D 75 49 03295          STA YG+3
4AE3: A9 00    03300          LDA #$00
4AE5: 8D 5A 49 03305          STA LASON
4AE8: 8D 5B 49 03310          STA LASON+1
4AEB: 8D 5C 49 03315          STA LASON+2
4AEE: 8D 5D 49 03320          STA LASON+3
```

```
4AF1: A9 01    03325          LDA #$01    ;LASERS SET ALIVE
4AF3: 8D 6E 49 03330          STA GALIVE
4AF6: 8D 6F 49 03335          STA GALIVE+1
4AF9: 8D 70 49 03340          STA GALIVE+2
4AFC: 8D 71 49 03345          STA GALIVE+3
               03350 *SETUP COLOR REGISTERS
4AFF: A9 00    03355          LDA #$00    ;BACKGROUND BLACK
4B01: 8D C8 02 03360          STA COLOR4
4B04: A9 24    03365          LDA #$24    ;TAN FOR PLAYFIELD#0
4B06: 8D C4 02 03370          STA COLOR0
4B09: A9 46    03375          LDA #$46    ;LT RED FOR PLAYFIELD #1
4B0B: 8D C5 02 03380          STA COLOR1
               03385 *MISC
4B0E: A9 08    03390          LDA #$08    ;SET FINE SCROLL HI SO SCREEN LEFT EDGE
4B10: 8D 04 D4 03395          STA HSCROL
4B13: A9 00    03400          LDA #$00
4B15: 8D 8E 49 03405          STA SLTIME
4B18: 8D 8F 49 03410          STA SLTIME1
4B1B: 8D 90 49 03415          STA SETIME
4B1E: 8D 92 49 03420          STA REZFLAG
4B21: 8D 93 49 03425          STA EXCOUNT
4B24: A9 00    03430          LDA #$00
4B26: 8D 05 D2 03435          STA AUDC3
4B29: 8D 08 D2 03440          STA AUDCTL  ;INITILIZE SOUND REGISTERS
4B2C: A9 03    03445          LDA #$03
4B2E: 8D 0F D2 03450          STA SKCTL
4B31: A9 30    03455          LDA #$30
4B33: 8D 1E 49 03460          STA TDELAY1
4B36: A9 50    03465          LDA #$50
4B38: 8D 1F 49 03470          STA TDELAY2
4B3B: A9 00    03475          LDA #$00
4B3D: 8D 2A 49 03480          STA TMIS0
4B40: 8D 2B 49 03485          STA TMIS1
4B43: 8D 2C 49 03490          STA TMIS2
4B46: 8D 82 49 03495          STA DELAY
4B49: 8D 1E D0 03500          STA HITCLR  ;CLEAR COLLISION REGISTERS
4B4C: 8D 18 49 03505          STA BOMBON  ;BOMB OFF
               03510 *PUT IN GROUND TARGETS
4B4F: A9 60    03515          LDA #$60    ;LASERS
4B51: 8D 3E 83 03520          STA $833E
4B54: 8D 71 86 03525          STA $8671
4B57: 8D 9E 86 03530          STA $869E
4B5A: 8D CB 86 03535          STA $86CB
4B5D: A9 61    03540          LDA #$61
4B5F: 8D 3F 83 03545          STA $833F
4B62: 8D 72 86 03550          STA $8672
4B65: 8D 9F 86 03555          STA $869F
4B68: 8D CC 86 03560          STA $86CC
4B6B: 20 D6 56 03565          JSR MBSUB
4B6E: A9 0A    03570          LDA #$0A
4B70: 8D 66 49 03575          STA LDELAY
4B73: 8D 67 49 03580          STA LDELAY+1
4B76: 8D 68 49 03585          STA LDELAY+2
4B79: 8D 69 49 03590          STA LDELAY+3
4B7C: A9 00    03595          LDA #$00    ;INTIALLY EASY
4B7E: 8D 94 49 03600          STA MEDIUM
4B81: 8D 95 49 03605          STA HARD
4B84: 8D 4C 46 03610          STA ENEMY1+$4C
4B87: 8D 91 46 03615          STA ENEMY1+$91
4B8A: 8D 9F 46 03620          STA ENEMY1+$9F
4B8D: 8D E5 46 03625          STA ENEMY1+$E5
```

```
4B90: 8D 1F 47 03630          STA ENEMY2+$1F
4B93: 8D 4C 47 03635          STA ENEMY2+$4C
4B96: 8D 79 47 03640          STA ENEMY2+$79
4B99: 8D B2 47 03645          STA ENEMY2+$B2
4B9C: 8D E5 47 03650          STA ENEMY2+$E5
4B9F: 8D 52 46 03655          STA ENEMY1+$52
4BA2: 8D 5E 46 03660          STA ENEMY1+$5E
4BA5: 8D 73 46 03665          STA ENEMY1+$73
4BA8: 8D 97 46 03670          STA ENEMY1+$97
4BAB: 8D B2 46 03675          STA ENEMY1+$B2
4BAE: 8D B8 46 03680          STA ENEMY1+$B8
4BB1: 8D EB 46 03685          STA ENEMY1+$EB
4BB4: 8D F1 46 03690          STA ENEMY1+$F1
4BB7: 8D 52 47 03695          STA ENEMY2+$52
4BBA: 8D 58 47 03700          STA ENEMY2+$58
4BBD: 8D 5E 47 03705          STA ENEMY2+$5E
4BC0: 8D 7F 47 03710          STA ENEMY2+$7F
4BC3: 8D 85 47 03715          STA ENEMY2+$85
4BC6: 8D C4 47 03720          STA ENEMY2+$C4
4BC9: 8D DF 47 03725          STA ENEMY2+$DF
4BCC: 8D EB 47 03730          STA ENEMY2+$EB
               03735 *ENABLE DISPLAY LIST INTERRUPT
4BCF: A9 C0    03740          LDA #$C0
4BD1: 8D 0E D4 03745          STA $D40E
4BD4: A9 71    03750          LDA #DLI
4BD6: 8D 00 02 03755          STA $200
4BD9: A9 52    03760          LDA /DLI
4BDB: 8D 01 02 03765          STA $201
               03770 *PUT TITLE INFO IN PLACE
4BDE: A0 00    03775 MKTITLE  LDY #$00
4BE0: B9 BD 48 03780 .1       LDA TITLE,Y
4BE3: 99 00 69 03785          STA INFO,Y
4BE6: C8       03790          INY
4BE7: C0 1D    03795          CPY #$1D
4BE9: 90 F5    03800          BLT .1
4BEB: AD 83 49 03805          LDA SHIPNUM
4BEE: 8D 1A 69 03810          STA INFO+26
4BF1: A9 33    03815          LDA #$33    ;PRINT 'SCORE'
4BF3: 8D 1D 69 03820          STA INFO+29
4BF6: A9 23    03825          LDA #$23
4BF8: 8D 1E 69 03830          STA INFO+30
4BFB: A9 2F    03835          LDA #$2F
4BFD: 8D 1F 69 03840          STA INFO+31
4C00: A9 32    03845          LDA #$32
4C02: 8D 20 69 03850          STA INFO+32
4C05: A9 25    03855          LDA #$25
4C07: 8D 21 69 03860          STA INFO+33
4C0A: A9 00    03865          LDA #$00
4C0C: 8D 22 69 03870          STA INFO+34
4C0F: A9 10    03875          LDA #$10    ;PRINT LAST ZERO DIGIT
4C11: 8D 27 69 03880          STA INFO+39
4C14: AD 7D 49 03885          LDA SR1OT
4C17: 8D 23 69 03890          STA INFO+35
4C1A: AD 7C 49 03895          LDA SR1000
4C1D: 8D 24 69 03900          STA INFO+36
4C20: AD 7B 49 03905          LDA SR100
4C23: 8D 25 69 03910          STA INFO+37
4C26: AD 7A 49 03915          LDA SR10
4C29: 8D 26 69 03920          STA INFO+38
4C2C: A9 80    03925          LDA #$80
4C2E: 85 14    03930          STA $14
```

```
4C30: A9 00    03935          LDA #$00
4C32: 85 13    03940          STA $13
4C34: 8D 8B 49 03945          STA T
4C37: AD 1F D0 03950 STOP     LDA CONSOL
4C3A: C9 06    03955          CMP #06     ; START KEY
4C3C: F0 5A    03960          BEQ MTITLE1
4C3E: A5 13    03965          LDA $13
4C40: F0 F5    03970          BEQ STOP
4C42: AD 8B 49 03975          LDA T
4C45: D0 97    03980          BNE MKTITLE
4C47: A0 00    03985 MYTITLE  LDY #$00
4C49: B9 E5 48 03990 .1       LDA TITLE1,Y
4C4C: 38       03995          SEC
4C4D: E9 20    04000          SBC #$20
4C4F: 99 00 69 04005          STA INFO,Y
4C52: C8       04010          INY
4C53: C0 14    04015          CPY #$14
4C55: 90 F2    04020          BLT .1
4C57: A9 00    04025          LDA #$00    ;STORE ' HIGH'
4C59: 8D 1D 69 04030          STA INFO+29
4C5C: A9 28    04035          LDA #$28
4C5E: 8D 1E 69 04040          STA INFO+30
4C61: A9 29    04045          LDA #$29
4C63: 8D 1F 69 04050          STA INFO+31
4C66: A9 27    04055          LDA #$27
4C68: 8D 20 69 04060          STA INFO+32
4C6B: A9 28    04065          LDA #$28
4C6D: 8D 21 69 04070          STA INFO+33
4C70: AD 81 49 04075          LDA HSR1OT
4C73: 8D 23 69 04080          STA INFO+35
4C76: AD 80 49 04085          LDA HSR1000
4C79: 8D 24 69 04090          STA INFO+36
4C7C: AD 7F 49 04095          LDA HSR100
4C7F: 8D 25 69 04100          STA INFO+37
4C82: AD 7E 49 04105          LDA HSR10
4C85: 8D 26 69 04110          STA INFO+38
4C88: A9 80    04115          LDA #$80
4C8A: 85 14    04120          STA $14
4C8C: A9 00    04125          LDA #$00
4C8E: 85 13    04130          STA $13
4C90: A9 01    04135          LDA #$01
4C92: 8D 8B 49 04140          STA T
4C95: 4C 37 4C 04145          JMP STOP
4C98: A0 00    04150 MTITLE1  LDY#$00
4C9A: B9 BD 48 04155 .1       LDA TITLE,Y
4C9D: 99 00 69 04160          STA INFO,Y
4CA0: C8       04165          INY
4CA1: C0 28    04170          CPY #$28
4CA3: 90 F5    04175          BLT .1
4CA5: AD 83 49 04180          LDA SHIPNUM
4CA8: 8D 1A 69 04185          STA INFO+26
               04190 *ZERO OUT SCORE
4CAB: A9 10    04195          LDA #$10
4CAD: 8D 7A 49 04200          STA SR10
4CB0: 8D 7B 49 04205          STA SR100
4CB3: 8D 7C 49 04210          STA SR1000
4CB6: 8D 7D 49 04215          STA SR10T
4CB9: A9 30    04220          LDA #$30    ;REG ENGINE SOUND
4CBB: 8D 04 D2 04225          STA AUDF3
4CBE: A9 86    04230          LDA #$86    ; DISTORTION 8,VOLUME 6
4CC0: 8D 05 D2 04235          STA AUDC3
```

```
                04240 *SET VBLANK
4CC3: A9 07     04245 FRAME     LDA #07
4CC5: A2 4E     04250           LDX /VBCODE ;HI BYTE VBLANK ROUTINE
4CC7: A0 BB     04255           LDY #VBCODE ;LO BYTE
4CC9: 20 5C E4  04260           JSR SETVBK
                04265 *MAIN LOOP CODE
4CCC: A9 00     04270 LOOPM     LDA #$00
4CCE: 8D 19 49  04275           STA VBFLAG
                04280 *FIRE GROUND LASER
4CD1: A2 03     04285 .2        LDX #$03
4CD3: BD 66 49  04290 FLASER    LDA LDELAY,X
4CD6: D0 08     04295           BNE .1
4CD8: A0 00     04300           LDY #$00
4CDA: 20 A7 54  04305           JSR LASER
4CDD: 4C E3 4C  04310           JMP CONT
4CE0: DE 66 49  04315 .1        DEC LDELAY,X
4CE3: CA        04320 CONT      DEX
4CE4: 10 ED     04325           BPL FLASER
                04330 *DETECT SHIP COLLISIONS
4CE6: AD 92 49  04335 COLLIDE   LDA REZFLAG ;DON'T ALLOW COLLISION DURING DEREZ
4CE9: D0 69     04340           BNE .4
4CEB: AD 04 D0  04345           LDA POPF    ;COLLISION WITH PLAYFIELD?
4CEE: F0 06     04350           BEQ .1
4CF0: 20 0A 56  04355           JSR XSHIP
4CF3: 4C 54 4D  04360           JMP .4
4CF6: AD 09 D0  04365 .1        LDA M1PL    ;COLLISION WITH MISSILE#1?
4CF9: C9 01     04370           CMP #$01
4CFB: D0 06     04375           BNE .2
4CFD: 20 0A 56  04380           JSR XSHIP
4D00: 4C 54 4D  04385           JMP .4
4D03: AD 0A D0  04390 .2        LDA M2PL    ;COLLISION WITH MISSILE#2?
4D06: C9 01     04395           CMP #$01
4D08: D0 06     04400           BNE .3
4D0A: 20 0A 56  04405           JSR XSHIP
4D0D: 4C 54 4D  04410           JMP .4
4D10: AD 0C D0  04415 .3        LDA POPL    ;COLLISION WITH PLAYER#1?
4D13: C9 02     04420           CMP #$02
4D15: D0 1E     04425           BNE .35
4D17: A9 E0     04430           LDA #$E0    ;REMOVE ALIEN#1
4D19: 8D 01 D0  04435           STA HPOSP1
4D1C: A9 20     04440           LDA #$20    ;TURN ON EXPLOSION SOUND
4D1E: 8D 90 49  04445           STA SETIME
4D21: A9 00     04450           LDA #$00
4D23: 8D 1C 49  04455           STA ONSCRN1
4D26: AD 20 49  04460           LDA NDELAY1
4D29: 8D 1E 49  04465           STA TDELAY1
4D2C: 20 C1 55  04470           JSR SCORE
4D2F: 20 0A 56  04475           JSR XSHIP
4D32: 4C 54 4D  04480           JMP .4
4D35: AD 0C D0  04485 .35       LDA POPL    ;COLLISION WITH PLAYER#2?
4D38: C9 04     04490           CMP #$04
4D3A: D0 18     04495           BNE .4
4D3C: A9 E0     04500           LDA #$E0    ;REMOVE ALIEN#2
4D3E: 8D 02 D0  04505           STA HPOSP2
4D41: A9 20     04510           LDA #$20    ;TURN ON EXPLOSION SOUND
4D43: 8D 90 49  04515           STA SETIME
4D46: A9 00     04520           LDA #$00
4D48: 8D 1D 49  04525           STA ONSCRN2
4D4B: AD 21 49  04530           LDA NDELAY2
4D4E: 8D 1F 49  04535           STA TDELAY2
4D51: 20 0A 56  04540           JSR XSHIP
```

```
4D54: AD 08 DO 04545 .4       LDA MOPL    ;COLLISION ALIEN#1 WITH MISSILE?
4D57: C9 02    04550          CMP #$02
4D59: DO 22    04555          BNE .5
4D5B: A9 EO    04560          LDA #$EO    ;REMOVE ALIEN#1
4D5D: 8D 01 DO 04565          STA HPOSP1
4D60: A9 20    04570          LDA #$20    ;TURN ON EXPLOSION SOUND
4D62: 8D 90 49 04575          STA SETIME
4D65: A9 00    04580          LDA #$00
4D67: 8D 1C 49 04585          STA ONSCRN1
4D6A: AD 20 49 04590          LDA NDELAY1
4D6D: 8D 1E 49 04595          STA TDELAY1
4D70: 20 AF 55 04600          JSR SCORE2
4D73: A9 00    04605          LDA #$00    ;REMOVE MISSILE
4D75: 8D 04 DO 04610          STA HPOSMO
4D78: A9 00    04615          LDA #$00    ;RESET TIMER
4D7A: 8D 2A 49 04620          STA TMISO
4D7D: AD 05 DO 04625 .5       LDA P1PF    ;ALIEN#1 COLLISION WITH PLAYFIELD?
4D80: FO 15    04630          BEQ .6
4D82: A9 EO    04635          LDA #$EO    ;REMOVE ALIEN#1
4D84: 8D 01 DO 04640          STA HPOSP1
4D87: A9 20    04645          LDA #$20    ;TURN ON EXPLOSION SOUND
4D89: 8D 90 49 04650          STA SETIME
4D8C: A9 00    04655          LDA #$00
4D8E: 8D 1C 49 04660          STA ONSCRN1
4D91: AD 20 49 04665          LDA NDELAY1
4D94: 8D 1E 49 04670          STA TDELAY1
4D97: AD 08 DO 04675 .6       LDA MOPL    ;COLLISION ALIEN#2 WITH SHIP MISSILE?
4D9A: C9 04    04680          CMP #$04
4D9C: DO 22    04685          BNE .7
4D9E: A9 EO    04690          LDA #$EO    ;REMOVE ALIEN#2
4DAO: 8D 02 DO 04695          STA HPOSP2
4DA3: A9 20    04700          LDA #$20    ;TURN ON EXPLOSION SOUND
4DA5: 8D 90 49 04705          STA SETIME
4DA8: A9 00    04710          LDA #$00
4DAA: 8D 1D 49 04715          STA ONSCRN2
4DAD: AD 21 49 04720          LDA NDELAY2
4DBO: 8D 1F 49 04725          STA TDELAY2
4DB3: 20 AF 55 04730          JSR SCORE2
4DB6: A9 00    04735          LDA #$00    ;REMOVE MISSILE
4DB8: 8D 04 DO 04740          STA HPOSMO
4DBB: A9 00    04745          LDA #$00
4DBD: 8D 2A 49 04750          STA TMISO
4DCO: AD 06 DO 04755 .7       LDA P2PF    ;ALIEN#2 COLLISION WITH PLAYFIELD?
4DC3: FO 15    04760          BEQ BCOL
4DC5: A9 EO    04765          LDA #$EO    ;REMOVE ALIEN #2
4DC7: 8D 02 DO 04770          STA HPOSP2
4DCA: A9 20    04775          LDA #$20    ;TURN ON EXPLOSION SOUND
4DCC: 8D 90 49 04780          STA SETIME
4DCF: A9 00    04785          LDA #$00
4DD1: 8D 1D 49 04790          STA ONSCRN2
4DD4: AD 21 49 04795          LDA NDELAY2
4DD7: 8D 1F 49 04800          STA TDELAY2
               04805 *TARGETS ARE PLAYFIELD#1 ;TERRAIN PLAYFIELD #0
4DDA: AD 07 DO 04810 BCOL     LDA P3PF    ;TEST BOMB COLLISION PLAYFIELD #0
4DDD: C9 01    04815          CMP #$01
4DDF: DO OD    04820          BNE .80
4DE1: A9 00    04825          LDA #$00    ;REMOVE BOMB
4DE3: 8D 03 DO 04830          STA HPOSP3
4DE6: A9 00    04835          LDA #$00
4DE8: 8D 18 49 04840          STA BOMBON
4DEB: 4C 08 4E 04845          JMP .82     ;JUST IN CASE MISSILE & BOMB ON SCREEN
```

```
4DEE: C9 02    04850 .80     CMP #$02    ;TEST BOMB AGAINST PLAYFIELD #1
4DF0: F0 03    04855         BEQ .81
4DF2: 4C 08 4E 04860         JMP .82
4DF5: AD 3A 49 04865 .81     LDA XB
4DF8: 8D 8A 49 04870         STA POS
4DFB: 20 0A 57 04875         JSR RTARGET
4DFE: A9 00    04880         LDA #$00    ;REMOVE BOMB
4E00: 8D 03 D0 04885         STA HPOSP3
4E03: A9 00    04890         LDA #$00
4E05: 8D 18 49 04895         STA BOMBON
4E08: AD 00 D0 04900 .82     LDA MOPF    ;TEST MISSILE COLLISION PLAYFIELD #0
4E0B: C9 01    04905         CMP #$01
4E0D: D0 0D    04910         BNE .83
4E0F: A9 00    04915         LDA #$00    ;REMOVE MISSILE
4E11: 8D 04 D0 04920         STA HPOSM0
4E14: A9 00    04925         LDA #$00
4E16: 8D 2A 49 04930         STA TMISC
4E19: 4C 36 4E 04935         JMP CC
4E1C: C9 02    04940 .83     CMP #$02    ;TEST MISSILE COLLISION PLAYFIELD #1
4E1E: F0 03    04945         BEQ .84
4E20: 4C 36 4E 04950         JMP CC
4E23: AD 3F 49 04955 .84     LDA XOM
4E26: 8D 8A 49 04960         STA POS
4E29: 20 0A 57 04965         JSR RTARGET
4E2C: A9 00    04970         LDA #$00    ;REMOVE MISSILE
4E2E: 8D 04 D0 04975         STA HPOSM0
4E31: A9 00    04980         LDA #$00
4E33: 8D 2A 49 04985         STA TMISC
4E36: 8D 1E D0 04990 CC      STA HITCLR  ;CLEAR COLLISIONS
4E39: A9 00    04995         LDA #$00    ;STOP ATTRACT MODE
4E3B: 85 4D    05000         STA $4D
4E3D: AD 95 49 05005 NASTY   LDA HARD    ;ON HARD SETTING?
4E40: D0 6C    05010         BNE FOREVER
4E42: AD 94 49 05015         LDA MEDIUM  ;ON MEDIUM SETTING?
4E45: D0 29    05020         BNE .1
4E47: AD 7B 49 05025         LDA SR100   ;AT 400 PTS
4E4A: C9 14    05030         CMP #$14    ;VALUE OFFSET BY #$10
4E4C: D0 22    05035         BNE .1
4E4E: A9 01    05040         LDA #$01    ;TURN ON SOME ALIEN GUNS
4E50: 8D 4C 46 05045         STA ENEMY1+$4C
4E53: 8D 91 46 05050         STA ENEMY1+$91
4E56: 8D 9F 46 05055         STA ENEMY1+$9F
4E59: 8D E5 46 05060         STA ENEMY1+$E5
4E5C: 8D 1F 47 05065         STA ENEMY2+$1F
4E5F: 8D 4C 47 05070         STA ENEMY2+$4C
4E62: 8D 79 47 05075         STA ENEMY2+$79
4E65: 8D B2 47 05080         STA ENEMY2+$B2
4E68: 8D E5 47 05085         STA ENEMY2+$E5
4E6B: A9 01    05090         LDA #$01
4E6D: 8D 94 49 05095         STA MEDIUM
4E70: AD 7C 49 05100 .1      LDA SR1000  ;AT 2000 PTS?
4E73: C9 12    05105         CMP #$12
4E75: D0 37    05110         BNE FOREVER
4E77: A9 01    05115         LDA #$01    ;TURN ON MORE ALIEN GUNS
4E79: 8D 52 46 05120         STA ENEMY1+$52
4E7C: 8D 5E 46 05125         STA ENEMY1+$5E
4E7F: 8D 73 46 05130         STA ENEMY1+$73
4E82: 8D 97 46 05135         STA ENEMY1+$97
4E85: 8D B2 46 05140         STA ENEMY1+$B2
4E88: 8D B8 46 05145         STA ENEMY1+$B8
4E8B: 8D EB 46 05150         STA ENEMY1+$EB
```

```
4E8E: 8D F1 46 05155          STA ENEMY1+$F1
4E91: 8D 52 47 05160          STA ENEMY2+$52
4E94: 8D 58 47 05165          STA ENEMY2+$58
4E97: 8D 5E 47 05170          STA ENEMY2+$5E
4E9A: 8D 7F 47 05175          STA ENEMY2+$7F
4E9D: 8D 85 47 05180          STA ENEMY2+$85
4EA0: 8D C4 47 05185          STA ENEMY2+$C4
4EA3: 8D DF 47 05190          STA ENEMY2+$DF
4EA6: 8D EB 47 05195          STA ENEMY2+$EB
4EA9: A9 01    05200          LDA #$01
4EAB: 8D 95 49 05205          STA HARD
4EAE: AD 19 49 05210 FOREVER  LDA VBFLAG
4EB1: C9 01    05215          CMP #$01
4EB3: D0 03    05220          BNE .1
4EB5: 4C CC 4C 05225          JMP LOOPM
4EB8: 4C AE 4E 05230 .1       JMP FOREVER
               05235 *TEST IF DELAY SET BY XSHIP
4EBB: A9 01    05240 VBCODE   LDA #$01
4EBD: 8D 19 49 05245          STA VBFLAG
4EC0: AD 82 49 05250          LDA DELAY
4EC3: F0 4E    05255          BEQ CHKSTK
4EC5: A5 13    05260          LDA $13
4EC7: F0 21    05265          BEQ REZ
4EC9: AD 83 49 05270          LDA SHIPNUM
4ECC: C9 10    05275          CMP #$10    ;COMPARE TO ZERO SHIPS
4ECE: D0 0B    05280          BNE .1
4ED0: A9 49    05285          LDA /START  ; SET RESET JMP VECTOR
4ED2: 85 03    05290          STA $03
4ED4: A9 C7    05295          LDA #START
4ED6: 85 02    05300          STA $02
4ED8: 4C 74 E4 05305          JMP $E474   ;SYSTEM RESET VECTOR
4EDB: A9 50    05310 .1       LDA #$50    ;PUT SHIP BACK
4EDD: 8D 00 D0 05315          STA HPOSP0
4EE0: A9 60    05320          LDA #$60
4EE2: 8D 0A 49 05325          STA YPM0
4EE5: A9 00    05330          LDA #$00
4EE7: 8D 82 49 05335          STA DELAY
4EEA: AD 92 49 05340 REZ      LDA REZFLAG
4EED: F0 24    05345          BEQ CHKSTK
4EEF: AD 93 49 05350          LDA EXCOUNT
4EF2: C9 30    05355          CMP #$30
4EF4: 90 17    05360          BLT .1
4EF6: A9 00    05365          LDA #$00    ;MOVE SHIP OFF SCREEN LEFT
4EF8: 8D 00 D0 05366          STA HPOSP0
4EFB: AD 93 49 05367          LDA EXCOUNT
4EFE: C9 31    05368          CMP #$31    ;DON'T RESET REZFLAG UNTIL NEXT CYCLE BECAUSE
4F00: D0 0B    05369          BNE .1      ;COULD GET COLLISION AGAINST PLAYFIELD IF ANY
4F02: A9 00    05370          LDA #$00    ;PIXELS LEFT IN SHIP SHAPE BEFORE MOVE
4F04: 8D 92 49 05380          STA REZFLAG
4F07: 8D 93 49 05385          STA EXCOUNT
4F0A: 4C 13 4F 05390          JMP CHKSTK
4F0D: 20 F6 57 05395 .1       JSR EXPLODE
4F10: EE 93 49 05400          INC EXCOUNT
               05405 *READ STICK
4F13: AD 78 02 05410 CHKSTK   LDA STICK
4F16: 29 01    05415          AND #$01    ;UP BIT?
4F18: D0 12    05420          BNE CHKDN
4F1A: CE 0A 49 05425          DEC YPM0
4F1D: CE 0A 49 05430          DEC YPM0
4F20: AD 0A 49 05435          LDA YPM0    ;TOP?
4F23: C9 30    05440          CMP #$30
```

```
4F25: B0 05    05445          BGE CHKDN
4F27: A9 30    05450          LDA #$30    ;CLIP AT TOP
4F29: 8D 0A 49 05455          STA YPM0
4F2C: AD 78 02 05460 CHKDN    LDA STICK
4F2F: 29 02    05465          AND #$02    ;DOWN BIT?
4F31: D0 12    05470          BNE CHKRT
4F33: EE 0A 49 05475          INC YPM0
4F36: EE 0A 49 05480          INC YPM0
4F39: AD 0A 49 05485          LDA YPM0    ;BOTTOM?
4F3C: C9 C0    05490          CMP #$C0
4F3E: 90 05    05495          BLT CHKRT
4F40: A9 C0    05500          LDA #$C0    ;CLIP AT BOTTOM
4F42: 8D 0A 49 05505          STA YPM0
4F45: AD 78 02 05510 CHKRT    LDA STICK
4F48: 29 08    05515          AND #$08    ;RIGHT BIT?
4F4A: D0 12    05520          BNE .1
4F4C: A9 01    05525          LDA #$01    ;FAST SPEED
4F4E: 8D 17 49 05530          STA SPEED
4F51: A9 20    05535          LDA #$20    ;FAST ENGINE SOUND
4F53: 8D 04 D2 05540          STA AUDF3
4F56: A9 86    05545          LDA #$86
4F58: 8D 05 D2 05550          STA AUDC3
4F5B: 4C 6D 4F 05555          JMP CHKFD
4F5E: A9 00    05560 .1       LDA #$00    ;REGULAR SPEED
4F60: 8D 17 49 05565          STA SPEED
4F63: A9 30    05570          LDA #$30    ;REG ENGINE SOUND
4F65: 8D 04 D2 05575          STA AUDF3
4F68: A9 86    05580          LDA #$86    ;
4F6A: 8D 05 D2 05585          STA AUDC3
4F6D: A9 00    05590 CHKFD    LDA #$00
4F6F: 8D 16 49 05595          STA BACK
4F72: AD 78 02 05600          LDA STICK
4F75: 29 04    05605          AND #$04    ;LEFT BIT
4F77: D0 05    05610          BNE .1
4F79: A9 01    05615          LDA #$01    ;YES STICK BACK
4F7B: 8D 16 49 05620          STA BACK
4F7E: AD 82 49 05625 .1       LDA DELAY   ;STOP SHIP SOUND IF IN DELAY
4F81: F0 05    05630          BEQ BUTTON
4F83: A9 00    05635          LDA #$00
4F85: 8D 05 D2 05640          STA AUDC3
               05645 *FIRE SHIP LASER
4F88: AD 82 49 05650 BUTTON   LDA DELAY   ;PREVENT MISSILE FIRING DURING DELAY
4F8B: D0 05    05655          BNE .15
4F8D: AD 84 02 05660          LDA STRIG0  ;BUTTON PRESSED=0
4F90: F0 08    05665          BEQ .1
4F92: AD 2A 49 05670 .15      LDA TMIS0   ;MISSILE ALREADY IN FLIGHT?
4F95: D0 34    05675          BNE .3
4F97: 4C EA 4F 05680          JMP BM
4F9A: AD 16 49 05685 .1       LDA BACK    ;STICK BACK?
4F9D: F0 03    05690          BEQ .2
4F9F: 4C 92 4F 05695          JMP .15
4FA2: AD 2A 49 05700 .2       LDA TMIS0   ;MISSILE ALREADY IN FLIGHT?
4FA5: D0 24    05705          BNE .3
4FA7: AD 09 49 05710          LDA XPM0    ;ADJUST LASER TO FIRE FROM SHIP'S NOSE
4FAA: 18       05715          CLC
4FAB: 69 10    05720          ADC #$10
4FAD: 8D 3F 49 05725          STA XOM
4FB0: AD 0A 49 05730          LDA YPM0
4FB3: 18       05735          CLC
4FB4: 69 05    05740          ADC #$05
4FB6: 8D 42 49 05745          STA YOM
```

```
4FB9: A2 00    05750          LDX #$00
4FBB: 20 76 54 05755          JSR MISSETO ;PLOT MISSILE
4FBE: A9 01    05760          LDA #$01    ;TURN ON MISSILE MOVING FLAG
4FC0: 8D 2A 49 05765          STA TMISO
4FC3: A9 01    05770          LDA #$01    ;TURN ON LASER SOUND
4FC5: 8D 8E 49 05775          STA SLTIME
4FC8: 4C EA 4F 05780          JMP BM
               05785 *MOVE MISSILE RIGHT
4FCB: EE 3F 49 05790 .3       INC XOM
4FCE: EE 3F 49 05795          INC XOM
4FD1: AD 3F 49 05800          LDA XOM
4FD4: C9 D8    05805          CMP #$D8
4FD6: B0 08    05810          BGE .4
4FD8: A2 00    05815          LDX #$00
4FDA: 20 76 54 05820          JSR MISSETO
4FDD: 4C EA 4F 05825          JMP BM
4FE0: A9 00    05830 .4       LDA #$00    ;REMOVE MISSILE TO LEFT
4FE2: 8D 2A 49 05835          STA TMISO
4FE5: A9 00    05840          LDA #$00
4FE7: 8D 04 D0 05845          STA HPOSMO
               05850 *DROP BOMB
4FEA: 20 30 55 05855 BM       JSR BOMB
4FED: AD 92 49 05860          LDA REZFLAG ;DON'T PLOT SHIP WHILE IN DEREZ
4FF0: D0 03    05865          BNE .1
4FF2: 20 9F 52 05870          JSR PLOTSET
               05875 *FINE SCROLL SCREEN
4FF5: AD 17 49 05880 .1       LDA SPEED
4FF8: F0 03    05885          BEQ SCROLL1
4FFA: EE 0E 49 05890 SCROLL   INC FS
4FFD: EE 0E 49 05895 SCROLL1  INC FS
5000: AD 0E 49 05900          LDA FS
5003: C9 08    05905          CMP #$08    ;GREATER 0-7 RANGE?
5005: B0 09    05910          BGE .1
5007: 38       05915          SEC
5008: E9 08    05920          SBC #$08
500A: 8D 0E 49 05925          STA FS
500D: EE 0D 49 05930          INC XS      ;UP ROUGH SCROLL
5010: 38       05935 .1       SEC
5011: A9 0F    05940          LDA #$0F    ;CORRECT 12 CLOCK CYCLES
5013: ED 0E 49 05945          SBC FS
5016: 8D 04 D4 05950          STA HSCROL
5019: AD 0D 49 05955          LDA XS      ;NEW XS
501C: C9 EC    05960          CMP #$EC    ;TEST WRAPAROUND @ 236
501E: 90 0F    05965          BLT .2
5020: A9 00    05970          LDA #$00
5022: 8D 0D 49 05975          STA XS
5025: AD 86 49 05980          LDA MBCOUNT ;IF ALL 7 BASES DESTROYED THEN REBUILD
5028: C9 07    05985          CMP #$07
502A: D0 03    05990          BNE .2
502C: 20 D6 56 05995          JSR MBSUB
502F: EA       06000 .2       NOP
               06005 *UPDATE DLIST 22 LO BYTES FOR EACH LMS
5030: A0 00    06010          LDY #$00    ;COUNTER
5032: AD 0D 49 06015 .4       LDA XS      ;POSITION AT SCREEN LEFT
5035: 99 08 94 06020          STA NDLIST+8,Y
5038: C8       06025          INY         ;LO BYTES ARE 3 APART
5039: C8       06030          INY
503A: C8       06035          INY
503B: C0 4B    06040          CPY #$4B    ;END OF LIST?
503D: D0 F3    06045          BNE .4      ;NEXT ELEMENT
503F: A9 E0    06050          LDA #$E0
```

```
5041: 8D F4 02 06055          STA 756
              06060 *PROGRAMABLE ALIENS
5044: AD 1E 49 06065 ATTACK   LDA TDELAY1 ;STILL IN DELAY?
5047: F0 15    06070          BEQ .05
5049: CE 1E 49 06075          DEC TDELAY1
504C: A9 00    06080          LDA #$00
504E: 8D 22 49 06085          STA TIMER1L
5051: AD 2B 49 06090          LDA TMIS1
5054: F0 05    06095          BEQ .08
5056: A2 01    06100          LDX #$01
5058: 20 49 53 06105          JSR MISSILE
505B: 4C 57 51 06110 .08      JMP EE
505E: AD 1C 49 06115 .05      LDA ONSCRN1 ;ALIEN #1 ON SCREEN?
5061: F0 03    06120          BEQ .07
5063: 4C C6 50 06125          JMP .1
5066: 20 E6 57 06130 .07      JSR CHANCE
5069: AE 87 49 06135          LDX LUCK    ;PLAYER #1,SHAPE#LUCK PATTERN
506C: BD 81 48 06140          LDA E1PT,X  ;SETUP TO READ DATA
506F: 85 F8    06145          STA E1L
5071: A9 46    06150          LDA /ENEMY1
5073: 85 F9    06155          STA E1H
5075: 20 E6 57 06160          JSR CHANCE  ;CHOOSE RANDOM SHAPE PLAYER#1
5078: AD 87 49 06165          LDA LUCK
507B: 8D 88 49 06170          STA RDSHP1
507E: A9 01    06175          LDA #$01    ;SET FLAG ON
5080: 8D 1C 49 06180          STA ONSCRN1
5083: A9 00    06185          LDA #$00    ;RESET TIMER
5085: 8D 22 49 06190          STA TIMER1L
5088: 8D 23 49 06195          STA TIMER1H
508B: A0 00    06200          LDY #$00
508D: B1 F8    06205          LDA (E1L),Y ;READ INITIAL STARTING VALUES
508F: 8D 38 49 06210          STA X1
5092: C8       06215          INY
5093: B1 F8    06220          LDA (E1L),Y
5095: 8D 3C 49 06225          STA Y1
5098: C8       06230          INY
5099: B1 F8    06235          LDA (E1L),Y
509B: 8D 20 49 06240          STA NDELAY1
509E: C8       06245          INY
509F: B1 F8    06250          LDA (E1L),Y
50A1: 8D 26 49 06255          STA TIME1L
50A4: C8       06260          INY
50A5: B1 F8    06265          LDA (E1L),Y
50A7: 8D 28 49 06270          STA TIME1H
50AA: C8       06275          INY
50AB: B1 F8    06280          LDA (E1L),Y
50AD: 8D 2E 49 06285          STA VX1
50B0: C8       06290          INY
50B1: B1 F8    06295          LDA (E1L),Y
50B3: 8D 32 49 06300          STA VY1
50B6: C8       06305          INY
50B7: B1 F8    06310          LDA (E1L),Y
50B9: 8D 46 49 06315          STA SHOOT1
50BC: C8       06320          INY         ;SKIP DIR
50BD: C8       06325          INY
50BE: 98       06330          TYA         ;SAVE Y REGISTER
50BF: 8D 12 49 06335          STA INDEX1
50C2: 4C 0A 51 06340          JMP .3
50C5: EA       06345          NOP
50C6: EE 22 49 06350 .1       INC TIMER1L
50C9: AD 22 49 06355          LDA TIMER1L ;CHECK IF TIMER HITS 256
```

```
50CC: D0 03    06360          BNE .2
50CE: EE 23 49 06365          INC TIMER1H
50D1: CD 26 49 06370 .2       CMP TIME1L  ;TIME TO READ NEXT SET INSTRUCTIONS?
50D4: D0 34    06375          BNE .3
50D6: AD 23 49 06380          LDA TIMER1H
50D9: CD 28 49 06385          CMP TIME1H
50DC: D0 2C    06390          BNE .3
50DE: AD 12 49 06395          LDA INDEX1  ;RESTORE Y REGISTER
50E1: A8       06400          TAY
50E2: B1 F8    06405          LDA (E1L),Y
50E4: 8D 26 49 06410          STA TIME1L
50E7: C8       06415          INY
50E8: B1 F8    06420          LDA (E1L),Y
50EA: 8D 28 49 06425          STA TIME1H
50ED: C8       06430          INY
50EE: B1 F8    06435          LDA (E1L),Y
50F0: 8D 2E 49 06440          STA VX1
50F3: C8       06445          INY
50F4: B1 F8    06450          LDA (E1L),Y
50F6: 8D 32 49 06455          STA VY1
50F9: C8       06460          INY
50FA: B1 F8    06465          LDA (E1L),Y
50FC: 8D 46 49 06470          STA SHOOT1
50FF: C8       06475          INY
5100: B1 F8    06480          LDA (E1L),Y
5102: 8D 51 49 06485          STA DIR1
5105: C8       06490          INY
5106: 98       06495          TYA         ;SAVE Y REGISTER
5107: 8D 12 49 06500          STA INDEX1
               06505 *MOVE ENEMY SHIP
510A: AD 38 49 06510 .3       LDA X1
510D: 18       06515          CLC
510E: 6D 2E 49 06520          ADC VX1
5111: 8D 38 49 06525          STA X1
5114: AD 17 49 06530          LDA SPEED ;MOVE ALIENS DOUBLE SPEED IF SHIP SPEEDING
5117: F0 03    06535          BEQ .35
5119: CE 38 49 06540          DEC X1
511C: AD 3C 49 06545 .35      LDA Y1
511F: 18       06550          CLC
5120: 6D 32 49 06555          ADC VY1
5123: 8D 3C 49 06560          STA Y1
5126: A2 01    06565          LDX #$01
5128: 20 49 53 06570          JSR MISSILE
               06575 *SHIP STILL ON SCREEN
512B: AD 38 49 06580 A1       LDA X1
512E: C9 30    06585          CMP #$30    ;X<48
5130: 90 15    06590          BLT .6
5132: C9 D1    06595          CMP #$D1    ;X>208
5134: B0 11    06600          BGE .6
5136: AD 3C 49 06605          LDA Y1
5139: C9 30    06610          CMP #$30    ;Y<48
513B: 90 0A    06615          BLT .6
513D: C9 E1    06620          CMP #$E1    ;Y>224
513F: B0 06    06625          BGE .6
5141: 20 C3 52 06630          JSR PLOTSET1
5144: 4C 57 51 06635          JMP EE
5147: A9 00    06640 .6       LDA #$00
5149: 8D 1C 49 06645          STA ONSCRN1
514C: AD 20 49 06650          LDA NDELAY1
514F: 8D 1E 49 06655          STA TDELAY1
5152: A9 E0    06660          LDA #$E0    ;REMOVE ALIEN FROM SCREEN
```

```
5154: 8D 01 D0 06665          STA HPOSP1
5157: EA       06670 EE       NOP
5158: AD 1F 49 06675 ATTACK2  LDA TDELAY2 ;STILL IN DELAY?
515B: F0 15    06680          BEQ .05
515D: CE 1F 49 06685          DEC TDELAY2
5160: A9 00    06690          LDA #$00
5162: 8D 24 49 06695          STA TIMER2L
5165: AD 2C 49 06700          LDA TMIS2
5168: F0 05    06705          BEQ .08
516A: A2 02    06710          LDX #$02
516C: 20 49 53 06715          JSR MISSILE
516F: 4C 6B 52 06720 .08      JMP EE2
5172: AD 1D 49 06725 .05      LDA ONSCRN2 ;ALIEN #1 ON SCREEN?
5175: F0 03    06730          BEQ .07
5177: 4C DA 51 06735          JMP .1
517A: 20 E6 57 06740 .07      JSR CHANCE
517D: AE 87 49 06745          LDX LUCK    ;PLAYER #2,SHAPE#LUCK PATTERN
5180: BD 86 48 06750          LDA E2PT,X  ;SETUP TO READ DATA
5183: 85 FA    06755          STA E2L
5185: A9 47    06760          LDA /ENEMY2
5187: 85 FB    06765          STA E2H
5189: A9 01    06770          LDA #$01    ;SET FLAG ON
518B: 8D 1D 49 06775          STA ONSCRN2
518E: 20 E6 57 06780          JSR CHANCE  ;CHOOSE RANDOM SHAPE PLAYER#2
5191: AD 87 49 06785          LDA LUCK
5194: 8D 89 49 06790          STA RDSHP2
5197: A9 00    06795          LDA #$00    ;RESET TIMER
5199: 8D 24 49 06800          STA TIMER2L
519C: 8D 25 49 06805          STA TIMER2H
519F: A0 00    06810          LDY #$00
51A1: B1 FA    06815          LDA (E2L),Y ;READ INITIAL STARTING VALUES
51A3: 8D 39 49 06820          STA X2
51A6: C8       06825          INY
51A7: B1 FA    06830          LDA (E2L),Y
51A9: 8D 3D 49 06835          STA Y2
51AC: C8       06840          INY
51AD: B1 FA    06845          LDA (E2L),Y
51AF: 8D 21 49 06850          STA NDELAY2
51B2: C8       06855          INY
51B3: B1 FA    06860          LDA (E2L),Y
51B5: 8D 27 49 06865          STA TIME2L
51B8: C8       06870          INY
51B9: B1 FA    06875          LDA (E2L),Y
51BB: 8D 29 49 06880          STA TIME2H
51BE: C8       06885          INY
51BF: B1 FA    06890          LDA (E2L),Y
51C1: 8D 2F 49 06895          STA VX2
51C4: C8       06900          INY
51C5: B1 FA    06905          LDA (E2L),Y
51C7: 8D 33 49 06910          STA VY2
51CA: C8       06915          INY
51CB: B1 FA    06920          LDA (E2L),Y
51CD: 8D 47 49 06925          STA SHOOT2
51D0: C8       06930          INY         ;SKIP DIR
51D1: C8       06935          INY
51D2: 98       06940          TYA         ;SAVE Y REGISTER
51D3: 8D 13 49 06945          STA INDEX2
51D6: 4C 1E 52 06950          JMP .3
51D9: EA       06955          NOP
51DA: EE 24 49 06960 .1       INC TIMER2L
51DD: AD 24 49 06965          LDA TIMER2L ;CHECK IF TIMER HITS 256
```

```
51E0: D0 03    06970          BNE .2
51E2: EE 25 49 06975          INC TIMER2H
51E5: CD 27 49 06980 .2       CMP TIME2L  ;TIME TO READ NEXT SET INSTRUCTIONS?
51E8: D0 34    06985          BNE .3
51EA: AD 25 49 06990          LDA TIMER2H
51ED: CD 29 49 06995          CMP TIME2H
51F0: D0 2C    07000          BNE .3
51F2: AD 13 49 07005          LDA INDEX2  ;RESTORE Y REGISTER
51F5: A8       07010          TAY
51F6: B1 FA    07015          LDA (E2L),Y
51F8: 8D 27 49 07020          STA TIME2L
51FB: C8       07025          INY
51FC: B1 FA    07030          LDA (E2L),Y
51FE: 8D 29 49 07035          STA TIME2H
5201: C8       07040          INY
5202: B1 FA    07045          LDA (E2L),Y
5204: 8D 2F 49 07050          STA VX2
5207: C8       07055          INY
5208: B1 FA    07060          LDA (E2L),Y
520A: 8D 33 49 07065          STA VY2
520D: C8       07070          INY
520E: B1 FA    07075          LDA (E2L),Y
5210: 8D 47 49 07080          STA SHOOT2
5213: C8       07085          INY
5214: B1 FA    07090          LDA (E2L),Y
5216: 8D 52 49 07095          STA DIR2
5219: C8       07100          INY
521A: 98       07105          TYA         ;SAVE Y REGISTER
521B: 8D 13 49 07110          STA INDEX2
               07115 *MOVE ENEMY SHIP
521E: AD 39 49 07120 .3       LDA X2
5221: 18       07125          CLC
5222: 6D 2F 49 07130          ADC VX2
5225: 8D 39 49 07135          STA X2
5228: AD 17 49 07140          LDA SPEED   ;MOVE ALIENS DOUBLE SPEED IF SHIP SPEEDING
522B: F0 03    07145          BEQ .35
522D: CE 39 49 07150          DEC X2
5230: AD 3D 49 07155 .35      LDA Y2
5233: 18       07160          CLC
5234: 6D 33 49 07165          ADC VY2
5237: 8D 3D 49 07170          STA Y2
523A: A2 02    07175          LDX #$02
523C: 20 49 53 07180          JSR MISSILE
               07185 *SHIP STILL ON SCREEN
523F: AD 39 49 07190 A2       LDA X2
5242: C9 30    07195          CMP #$30    ;X<48
5244: 90 15    07200          BLT .6
5246: C9 D1    07205          CMP #$D1    ;X>208
5248: B0 11    07210          BGE .6
524A: AD 3D 49 07215          LDA Y2
524D: C9 30    07220          CMP #$30    ;Y<48
524F: 90 0A    07225          BLT .6
5251: C9 E1    07230          CMP #$E1    ;Y>224
5253: B0 06    07235          BGE .6
5255: 20 F1 52 07240          JSR PLOTSET2
5258: 4C 6B 52 07245          JMP EE2
525B: A9 00    07250 .6       LDA #$00
525D: 8D 1D 49 07255          STA ONSCRN2
5260: AD 21 49 07260          LDA NDELAY2
5263: 8D 1F 49 07265          STA TDELAY2
5266: A9 E0    07270          LDA #$E0    ;REMOVE ALIEN FROM SCREEN
```

```
5268: 8D 02 D0 07275          STA HPOSP2
526B: 20 5C 58 07280 EE2      JSR SOUND
526E: 4C 62 E4 07285 PAST     JMP XITVBK
              02260           .IN "D:SCROLL1C"
              07500 *PART 3 OF SCROLLING GAME
              07505 *DISPLAY LIST INTERUPT ROUTINE
5271: 48      07510 DLI       PHA
5272: A9 90   07515           LDA #$90      ;HI BYTE OF CUSTOM SET
5274: 8D 0A D4 07520          STA WSYNC
5277: 8D 09 D4 07525          STA CHBASE
527A: 68      07530           PLA
527B: 40      07535           RTI
              07540 *PUT SHAPE IN P/M AREA
527C: A0 00   07545 PLOT      LDY #$00      ;COUNTER
527E: 91 F2   07550           STA (SHPML),Y ;PUT IN P/M AREA
5280: A9 00   07555           LDA #$00      ;NEED 0 TO ERASE EACH TIME
5282: 91 F4   07560 .1        STA (SHPMOL),Y ;ERASE OLD SHAPE FIRST
5284: C8      07565           INY
5285: C0 08   07570           CPY #$08
5287: 90 F9   07575           BLT .1
5289: A0 00   07580           LDY #$00
528B: B1 F0   07585 .2        LDA (SHPL),Y  ;GET BYTE FROM PROPER SHAPE TABLE
528D: 91 F2   07590           STA (SHPML),Y ;PUT IN P/M AREA
528F: C8      07595           INY
5290: C0 08   07600           CPY #$08
5292: 90 F7   07605           BLT .2
5294: A5 F2   07610           LDA SHPML   ;TRANSFER NEW P/M POS TO OLD POS
5296: 9D 48 49 07615          STA TEMPL,X
5299: A5 F3   07620           LDA SHPMH
529B: 9D 4C 49 07625          STA TEMPH,X
529E: 60      07630           RTS
529F: AD 0A 49 07635 PLOTSET  LDA YPMO    ;CORRECTED YPOS
52A2: 85 F2   07640           STA SHPML
52A4: A9 88   07645           LDA /PDATA
52A6: 18      07650           CLC
52A7: 69 04   07655           ADC #$04    ;PLAYER0 IS 1K BEYOND START
52A9: 85 F3   07660           STA SHPMH
52AB: A9 48   07665           LDA /SHIP
52AD: 85 F1   07670           STA SHPH
52AF: A9 4C   07675           LDA #SHIP
52B1: 85 F0   07680           STA SHPL
52B3: A2 00   07685           LDX #$00
52B5: BD 48 49 07690          LDA TEMPL,X
52B8: 85 F4   07695           STA SHPMOL
52BA: BD 4C 49 07700          LDA TEMPH,X
52BD: 85 F5   07705           STA SHPMOH
52BF: 20 7C 52 07710          JSR PLOT
52C2: 60      07715           RTS
              07720 *PLOTSET1 SUBROUTINE
52C3: AD 38 49 07725 PLOTSET1 LDA X1
52C6: 8D 01 D0 07730          STA HPOSP1
52C9: AD 3C 49 07735          LDA Y1
52CC: 85 F2   07740           STA SHPML
52CE: A9 88   07745           LDA /PDATA
52D0: 18      07750           CLC
52D1: 69 05   07755           ADC #$05    ;PLAYER#1 IS 1.25K BEYOND START
52D3: 85 F3   07760           STA SHPMH
52D5: A9 48   07765           LDA /ALIEN
52D7: 85 F1   07770           STA SHPH
52D9: AE 88 49 07775          LDX RDSHP1
52DC: BD 7C 48 07780          LDA ALIENPT,X
```

```
52DF: 85 F0     07785           STA SHPL
52E1: A2 01     07790           LDX #$01
52E3: BD 48 49  07795           LDA TEMPL,X
52E6: 85 F4     07800           STA SHPMOL
52E8: BD 4C 49  07805           LDA TEMPH,X
52EB: 85 F5     07810           STA SHPMOH
52ED: 20 7C 52  07815           JSR PLOT
52F0: 60        07820           RTS
52F1: AD 39 49  07825 PLOTSET2  LDA X2
52F4: 8D 02 D0  07830           STA HPOSP2
52F7: AD 3D 49  07835           LDA Y2
52FA: 85 F2     07840           STA SHPML
52FC: A9 88     07845           LDA /PDATA
52FE: 18        07850           CLC
52FF: 69 06     07855           ADC #$06    ;PLAYER#1 IS 1.5K BEYOND START
5301: 85 F3     07860           STA SHPMH
5303: A9 48     07865           LDA /ALIEN
5305: 85 F1     07870           STA SHPH
5307: AE 89 49  07875           LDX RDSHP2
530A: BD 7C 48  07880           LDA ALIENPT,X
530D: 85 F0     07885           STA SHPL
530F: A2 02     07890           LDX #$02
5311: BD 48 49  07895           LDA TEMPL,X
5314: 85 F4     07900           STA SHPMOL
5316: BD 4C 49  07905           LDA TEMPH,X
5319: 85 F5     07910           STA SHPMOH
531B: 20 7C 52  07915           JSR PLOT
531E: 60        07920           RTS
531F: AD 3A 49  07925 PLOTSET3  LDA XB
5322: 8D 03 D0  07930           STA HPOSP3
5325: AD 3E 49  07935           LDA YB
5328: 85 F2     07940           STA SHPML
532A: A9 88     07945           LDA /PDATA
532C: 18        07950           CLC
532D: 69 07     07955           ADC #$07    ;PLAYER#1 IS 1.75K BEYOND START
532F: 85 F3     07960           STA SHPMH
5331: A9 48     07965           LDA /BOMBSH
5333: 85 F1     07970           STA SHPH
5335: A9 97     07975           LDA #BOMBSH
5337: 85 F0     07980           STA SHPL
5339: A2 03     07985           LDX #$03
533B: BD 48 49  07990           LDA TEMPL,X
533E: 85 F4     07995           STA SHPMOL
5340: BD 4C 49  08000           LDA TEMPH,X
5343: 85 F5     08005           STA SHPMOH
5345: 20 7C 52  08010           JSR PLOT
5348: 60        08015           RTS
                08020 *MISSILE SUBROUTINE
5349: BD 45 49  08025 MISSILE   LDA SHOOTO,X
534C: F0 30     08030           BEQ .08
534E: BD 2A 49  08035           LDA TMISO,X
5351: D0 33     08040           BNE .1
5353: FE 2A 49  08045           INC TMISO,X ;INCREMENT TIMER
5356: BD 50 49  08050           LDA DIRO,X  ;SAVE TABLE DIRECTION
5359: 9D 53 49  08055           STA MDIRO,X
                08060 *ERASE OLD MISSILE & PUT AT SHIP
535C: BD 3B 49  08065           LDA YO,X
535F: 18        08070           CLC
5360: 69 04     08075           ADC #$04    ;CORRECT TO SHIP CENTER
5362: 9D 42 49  08080           STA YOM,X
5365: BD 37 49  08085           LDA XO,X
```

```
5368: 18       08090          CLC
5369: 69 04    08095          ADC #$04
536B: 9D 3F 49 08100          STA XOM,X
536E: E0 02    08105          CPX #$02
5370: F0 06    08110          BEQ .05
5372: 20 14 54 08115          JSR MISSET1
5375: 4C EF 53 08120          JMP DD
5378: 20 45 54 08125 .05      JSR MISSET2
537B: 4C EF 53 08130          JMP DD
537E: BD 2A 49 08135 .08      LDA TMISO,X
5381: D0 03    08140          BNE .1
5383: 4C EF 53 08145          JMP DD
5386: BD 2A 49 08150 .1       LDA TMISO,X
5389: C9 1E    08155          CMP #$1E    ;MISSILE CAN ONLY MOVE 30 CYCLES
538B: B0 4B    08160          BGE E
               08165 *MOVE MISSILE IN PROPER DIRECTION
538D: BC 53 49 08170 .2       LDY MDIRO,X
5390: B9 9F 48 08175          LDA VMX,Y
5393: 0A       08180          ASL         ;DOUBLE VELOCITY
5394: 18       08185          CLC
5395: 7D 3F 49 08190          ADC XOM,X
5398: 9D 3F 49 08195          STA XOM,X
539B: B9 A7 48 08200          LDA VMY,Y
539E: 0A       08205          ASL         ;DOUBLE VELOCITY
539F: 18       08210          CLC
53A0: 7D 42 49 08215          ADC YOM,X
53A3: 9D 42 49 08220          STA YOM,X
               08225 *HAS MISSILE HIT SCREEN EDGE
53A6: C9 30    08230          CMP #$30
53A8: B0 03    08235          BGE .23
53AA: 4C D8 53 08240          JMP E
53AD: C9 D8    08245 .23      CMP #$D8
53AF: 90 03    08250          BLT .24
53B1: 4C D8 53 08255          JMP E
53B4: BD 3F 49 08260 .24      LDA XOM,X
53B7: C9 30    08265          CMP #$30
53B9: B0 03    08270          BGE .25
53BB: 4C D8 53 08275          JMP E
53BE: C9 D0    08280 .25      CMP #$D0
53C0: 90 03    08285          BLT .26
53C2: 4C D8 53 08290          JMP E
53C5: FE 2A 49 08295 .26      INC TMISO,X
               08300 *ERASE & REPLOT MISSILE
53C8: E0 02    08305 ERASE    CPX #$02
53CA: F0 06    08310          BEQ .28
53CC: 20 14 54 08315          JSR MISSET1
53CF: 4C EF 53 08320          JMP DD
53D2: 20 45 54 08325 .28      JSR MISSET2
53D5: 4C EF 53 08330          JMP DD
               08335 *ERASE MISSILE OFF SCREEN
               08340 *MISSILES BOMBS & SHIPS PUT ON FAR LEFT TO PREVENT COLLISIONS
               08345 *WITH ALIENS & THEIR MISSILES PUT AT FAR RIGHT
53D8: A9 E8    08350 E        LDA #$E8
53DA: 9D 3F 49 08355          STA XOM,X   ;PLOT OFF SCREEN
53DD: E0 02    08360          CPX #$02
53DF: F0 06    08365          BEQ .31
53E1: 20 14 54 08370          JSR MISSET1
53E4: 4C EA 53 08375          JMP .7
53E7: 20 45 54 08380 .31      JSR MISSET2
53EA: A9 00    08385 .7       LDA #$00
53EC: 9D 2A 49 08390          STA TMISO,X
```

```
53EF: 60        08395 DD       RTS
                08400 *PUT MISSILE SHAPE IN P/M AREA SUBROUTINE
53F0: A0 00     08405 MPLOT    LDY #$00
53F2: B1 F4     08410 .1       LDA (SHPMOL),Y
53F4: 2D 56 49  08415          AND MASK
53F7: 91 F4     08420          STA (SHPMOL),Y
53F9: C8        08425          INY
53FA: C0 02     08430          CPY #$02
53FC: 90 F4     08435          BLT .1
53FE: A0 00     08440          LDY #$00
5400: B1 F0     08445 .2       LDA (SHPL),Y
5402: 11 F2     08450          ORA (SHPML),Y
5404: 91 F2     08455          STA (SHPML),Y
5406: C8        08460          INY
5407: C0 02     08465          CPY #$02
5409: 90 F5     08470          BLT .2
540B: A5 F2     08475          LDA SHPML
540D: 85 F4     08480          STA SHPMOL
540F: A5 F3     08485          LDA SHPMH
5411: 85 F5     08490          STA SHPMOH
5413: 60        08495          RTS
                08500 *SETUP TO PLOT MISSILE 1
5414: AD 43 49  08505 MISSET1  LDA Y1M ;MISSILE POSITION CORRECTED
5417: 85 F2     08510          STA SHPML
5419: A9 88     08515          LDA /PDATA
541B: 18        08520          CLC
541C: 69 03     08525          ADC #$03    ;MISSILES .75K BEYOND START
541E: 85 F3     08530          STA SHPMH
5420: 85 F5     08535          STA SHPMOH
5422: BD 93 48  08540          LDA MISLO,X ;POINTER TO CORRECT MISSILE SHAPE
5425: 85 F0     08545          STA SHPL
5427: A9 48     08550          LDA /MSHAPE ;HI BYTE BOTH P/M SHAPES SAME
5429: 85 F1     08555          STA SHPH
542B: AD 10 49  08560          LDA YMISOLD1
542E: 85 F4     08565          STA SHPMOL
5430: AD 58 49  08570          LDA MASK1
5433: 8D 56 49  08575          STA MASK
5436: 20 F0 53  08580          JSR MPLOT
5439: A5 F4     08585          LDA SHPMOL
543B: 8D 10 49  08590          STA YMISOLD1
543E: AD 40 49  08595          LDA X1M
5441: 8D 05 D0  08600          STA HPOSM1  ;MISSILE 1 HORIZ POS
5444: 60        08605          RTS
                08610 *SETUP TO PLOT MISSILE 2
5445: AD 44 49  08615 MISSET2  LDA Y2M ;MISSILE POSITION CORRECTED
5448: 85 F2     08620          STA SHPML
544A: A9 88     08625          LDA /PDATA
544C: 18        08630          CLC
544D: 69 03     08635          ADC #$03    ;MISSILES .75K BEYOND START
544F: 85 F3     08640          STA SHPMH
5451: 85 F5     08645          STA SHPMOH
5453: BD 93 48  08650          LDA MISLO,X ;POINTER TO CORRECT MISSILE SHAPE
5456: 85 F0     08655          STA SHPL
5458: A9 48     08660          LDA /MSHAPE ;HI BYTE BOTH P/M SHAPES SAME
545A: 85 F1     08665          STA SHPH
545C: AD 11 49  08670          LDA YMISOLD2
545F: 85 F4     08675          STA SHPMOL
5461: AD 59 49  08680          LDA MASK2
5464: 8D 56 49  08685          STA MASK
5467: 20 F0 53  08690          JSR MPLOT
546A: A5 F4     08695          LDA SHPMOL
```

```
546C: 8D 11 49 08700          STA YMISOLD2
546F: AD 41 49 08705          LDA X2M
5472: 8D 06 DO 08710          STA HPOSM2  ;MISSILE 2 HORIZ POS
5475: 60       08715          RTS
               08720 *SETUP TO PLOT MISSILE 0
5476: AD 42 49 08725 MISSETO  LDA YOM ;MISSILE POSITION CORRECTED
5479: 85 F2    08730          STA SHPML
547B: A9 88    08735          LDA /PDATA
547D: 18       08740          CLC
547E: 69 03    08745          ADC #$03    ;MISSILES .75K BEYOND START
5480: 85 F3    08750          STA SHPMH
5482: 85 F5    08755          STA SHPMOH
5484: BD 93 48 08760          LDA MISLO,X ;POINTER TO CORRECT MISSILE SHAPE
5487: 85 F0    08765          STA SHPL
5489: A9 48    08770          LDA /MSHAPE ;HI BYTE BOTH P/M SHAPES SAME
548B: 85 F1    08775          STA SHPH
548D: AD 0F 49 08780          LDA YMISOLDO
5490: 85 F4    08785          STA SHPMOL
5492: AD 57 49 08790          LDA MASKO
5495: 8D 56 49 08795          STA MASK
5498: 20 F0 53 08800          JSR MPLOT
549B: A5 F4    08805          LDA SHPMOL
549D: 8D 0F 49 08810          STA YMISOLDO
54A0: AD 3F 49 08815          LDA XOM
54A3: 8D 04 DO 08820          STA HPOSMO  ;MISSILE 0 HORIZ POS
54A6: 60       08825          RTS
54A7: BD 5A 49 08830 LASER    LDA LASON,X
54AA: DO 38    08835          BNE .1
54AC: BD 6E 49 08840          LDA GALIVE,X
54AF: DO 03    08845          BNE .11
54B1: 4C 2F 55 08850          JMP EL
54B4: A9 01    08855 .11      LDA #$01    ;TURN LASER ON
54B6: 9D 5A 49 08860          STA LASON,X
54B9: A9 70    08865          LDA #$70
54BB: 18       08870          CLC
54BC: 7D 72 49 08875          ADC YG,X
54BF: 9D 62 49 08880          STA LPOSH,X
54C2: DE 62 49 08885          DEC LPOSH,X ;BEAM STARTS JUST ABOVE GUN
54C5: BD 76 49 08890          LDA XG,X
54C8: 9D 5E 49 08895          STA LPOSL,X
54CB: DE 5E 49 08900          DEC LPOSL,X
54CE: A9 03    08905          LDA #$03    ;DELAY BETWEEN MOVENTS
54D0: 9D 6A 49 08910          STA LCOUNT,X
54D3: BD 62 49 08915          LDA LPOSH,X
54D6: 85 F7    08920          STA GROUNDH
54D8: BD 5E 49 08925          LDA LPOSL,X
54DB: 85 F6    08930          STA GROUNDL
54DD: A9 62    08935          LDA #$62    ;PLOT INITIAL POSITION LASER
54DF: 91 F6    08940          STA (GROUNDL),Y
54E1: 4C 2F 55 08945          JMP EL
54E4: DE 6A 49 08950 .1       DEC LCOUNT,X
54E7: BD 6A 49 08955          LDA LCOUNT,X;MOVE BEAM EVERY 3RD FRAME
54EA: DO 43    08960          BNE EL
54EC: BD 62 49 08965          LDA LPOSH,X ;AT TOP OF SCREEN?
54EF: C9 70    08970          CMP #$70
54F1: DO 1B    08975          BNE .2
54F3: BD 62 49 08980          LDA LPOSH,X
54F6: 85 F7    08985          STA GROUNDH
54F8: BD 5E 49 08990          LDA LPOSL,X
54FB: 85 F6    08995          STA GROUNDL
54FD: A9 00    09000          LDA #$00    ;ERASE LASER
```

```
54FF: 91 F6    09005          STA (GROUNDL),Y
5501: A9 00    09010          LDA #$00    ;TURN LASER OFF
5503: 9D 5A 49 09015          STA LASON,X
5506: A9 50    09020          LDA #$50    ;DELAY BETWEEN FIRING
5508: 9D 66 49 09025          STA LDELAY,X
550B: 4C 2F 55 09030          JMP EL
550E: BD 62 49 09035 .2       LDA LPOSH,X
5511: 85 F7    09040          STA GROUNDH
5513: BD 5E 49 09045          LDA LPOSL,X
5516: 85 F6    09050          STA GROUNDL
5518: A9 00    09055          LDA #$00    ;ERASE OLD POSITION LASER
551A: 91 F6    09060          STA (GROUNDL),Y
551C: C6 F7    09065          DEC GROUNDH
551E: DE 62 49 09070          DEC LPOSH,X
5521: C6 F6    09075          DEC GROUNDL
5523: DE 5E 49 09080          DEC LPOSL,X
5526: A9 62    09085          LDA #$62
5528: 91 F6    09090          STA (GROUNDL),Y ;PLOT NEW POSITION
552A: A9 03    09095          LDA #$03
552C: 9D 6A 49 09100          STA LCOUNT,X
552F: 60       09105 EL       RTS
               09110 *BOMB SUBROUTINE
5530: AD 82 49 09115 BOMB     LDA DELAY   ;PREVENT BOMB DROP DURING DELAY
5533: D0 05    09120          BNE .1
5535: AD 84 02 09125          LDA STRIGO  ;BUTTON PRESSED=0
5538: F0 08    09130          BEQ .2
553A: AD 18 49 09135 .1       LDA BOMBON
553D: D0 34    09140          BNE .4
553F: 4C 9C 55 09145          JMP EBM
5542: AD 16 49 09150 .2       LDA BACK
5545: F0 F3    09155          BEQ .1
5547: AD 18 49 09160 .3       LDA BOMBON
554A: D0 27    09165          BNE .4
               09170 *DROP BOMB INITALLY
554C: AD 0A 49 09175          LDA YPMO    ;CENTER BOMB UNDER SHIP
554F: 18       09180          CLC
5550: 69 0A    09185          ADC #$0A
5552: 8D 3E 49 09190          STA YB
5555: AD 09 49 09195          LDA XPMO
5558: 18       09200          CLC
5559: 69 05    09205          ADC #$05
555B: 8D 3A 49 09210          STA XB
555E: A9 00    09215          LDA #$00
5560: 8D 34 49 09220          STA VY3
5563: 8D 35 49 09225          STA VTEMP   ;INITILIZE ACCELERATION
5566: A9 01    09230          LDA #$01
5568: 8D 18 49 09235          STA BOMBON
556B: A2 03    09240          LDX #$03
556D: 20 1F 53 09245          JSR PLOTSET3
5570: 4C 9C 55 09250          JMP EBM
               09255 *CALCULATE & PLOT NEW BOMB POSITION
5573: EE 35 49 09260 .4       INC VTEMP
5576: AD 35 49 09265          LDA VTEMP
5579: 4A       09270          LSR         ;DIVIDE BY 4
557A: 4A       09275          LSR
557B: C9 03    09280          CMP #$03
557D: 90 02    09285          BLT .45
557F: A9 03    09290          LDA #$03    ;CLIP TO 3
5581: 8D 34 49 09295 .45      STA VY3     ;NEW VY3
5584: AD 3E 49 09300          LDA YB
5587: 18       09305          CLC
```

```
5588: 6D 34 49 09310          ADC VY3
558B: 8D 3E 49 09315          STA YB
558E: AD 3A 49 09320          LDA XB
5591: 18      09325           CLC
5592: 69 01   09330           ADC #$01    ;FORWARD VEL=1
5594: 8D 3A 49 09335          STA XB
5597: A2 03   09340           LDX #$03
5599: 20 1F 53 09345          JSR PLOTSET3
559C: 60      09350 EBM       RTS
              09355 *SCORE SUBROUTINE
559D: EE 7A 49 09360 SCORE3   INC SR10
55A0: AD 7A 49 09365          LDA SR10
55A3: C9 1A   09370           CMP #$1A    ;ADD 16 FOR INTERNAL VALUES
55A5: 90 08   09375           BLT SCORE2
55A7: EE 7B 49 09380          INC SR100
55AA: A9 10   09385           LDA #$10    ;CHARACTER 0 IS INTERNAL $10
55AC: 8D 7A 49 09390          STA SR10
55AF: EE 7A 49 09395 SCORE2   INC SR10
55B2: AD 7A 49 09400          LDA SR10
55B5: C9 1A   09405           CMP #$1A
55B7: 90 08   09410           BLT SCORE
55B9: EE 7B 49 09415          INC SR100
55BC: A9 10   09420           LDA #$10
55BE: 8D 7A 49 09425          STA SR10
55C1: EE 7A 49 09430 SCORE    INC SR10
55C4: AD 7A 49 09435          LDA SR10
55C7: C9 1A   09440           CMP #$1A
55C9: 90 08   09445           BLT .1
55CB: EE 7B 49 09450          INC SR100
55CE: A9 10   09455           LDA #$10
55D0: 8D 7A 49 09460          STA SR10
55D3: AD 7B 49 09465 .1       LDA SR100
55D6: C9 1A   09470           CMP #$1A
55D8: 90 08   09475           BLT .2
55DA: EE 7C 49 09480          INC SR1000
55DD: A9 10   09485           LDA #$10
55DF: 8D 7B 49 09490          STA SR100
55E2: AD 7C 49 09495 .2       LDA SR1000
55E5: C9 1A   09500           CMP #$1A
55E7: 90 08   09505           BLT .3
55E9: EE 7D 49 09510          INC SR10T
55EC: A9 10   09515           LDA #$10
55EE: 8D 7C 49 09520          STA SR1000
              09525 *PLACE VALUES IN SCORE LINE
55F1: AD 7A 49 09530 .3       LDA SR10
55F4: 8D 26 69 09535          STA INFO+38
55F7: AD 7B 49 09540          LDA SR100
55FA: 8D 25 69 09545          STA INFO+37
55FD: AD 7C 49 09550          LDA SR1000
5600: 8D 24 69 09555          STA INFO+36
5603: AD 7D 49 09560          LDA SR10T
5606: 8D 23 69 09565          STA INFO+35
5609: 60      09570           RTS
              09575 *ERASE SHIP SUBROUTINE
560A: A9 00   09580 XSHIP     LDA #$00
560C: 8D 93 49 09585          STA EXCOUNT
560F: A9 01   09590           LDA #$01    ;TURN DEREZ EXPLOSION ON
5611: 8D 92 49 09595          STA REZFLAG
5614: A9 40   09600           LDA #$40    ;TURN ON SHIP EXPLOSION SOUND
5616: 8D 91 49 09605          STA SEXTIME
5619: CE 83 49 09610          DEC SHIPNUM
```

```
561C: AD 83 49 09615          LDA SHIPNUM
561F: 8D 1A 69 09620          STA INFO+26
5622: C9 10    09625          CMP #$10     ;0 SHIPS?
5624: F0 18    09630          BEQ EGAME
5626: A9 01    09635          LDA #$01     ;TURN ON DELAY
5628: 8D 82 49 09640          STA DELAY
562B: A9 00    09645          LDA #$00     ;4 SECOND DELAY
562D: 85 14    09650          STA $14
562F: A9 00    09655          LDA #$00
5631: 85 13    09660          STA $13
5633: A9 F0    09665          LDA #$F0     ;4 SECOND DELAY
5635: 8D 20 49 09670          STA NDELAY1
5638: A9 D0    09675          LDA #$D0     ;3+ SECOND DELAY
563A: 8D 21 49 09680          STA NDELAY2
563D: 60       09685          RTS
563E: AD 0D 49 09690 EGAME    LDA XS       ;END GAME BY WRITEING "END GAME"
5641: 69 14    09695          ADC #$14
5643: 8D 8C 49 09700          STA OVER
5646: AA       09705          TAX
5647: A0 00    09710          LDY #$00
5649: B9 00 49 09715 .1       LDA GOVER,Y
564C: 9D 00 7B 09720          STA $7B00,X
564F: E8       09725          INX
5650: C8       09730          INY
5651: C0 09    09735          CPY #$09
5653: 90 F4    09740          BLT .1
               09745 *TEST IF NEW HIGH SCORE
5655: 38       09750 HIGH     SEC
5656: AD 7D 49 09755          LDA SR10T
5659: ED 81 49 09760          SBC HSR10T
565C: F0 04    09762          BEQ .1       ;TEST NEXT DIGIT IF 0
565E: 10 25    09765          BPL THIGH    ;UPDATE HI SCORE
5660: 30 3B    09770          BMI ED       ;SCORE<HI SCORE - EXIT
5662: 38       09775 .1       SEC
5663: AD 7C 49 09780          LDA SR1000
5666: ED 80 49 09785          SBC HSR1000
5669: F0 04    09787          BEQ .2
566B: 10 18    09790          BPL THIGH
566D: 30 2E    09795          BMI ED
566F: 38       09800 .2       SEC
5670: AD 7B 49 09805          LDA SR100
5673: ED 7F 49 09810          SBC HSR100
5676: F0 04    09812          BEQ .3
5678: 10 0B    09815          BPL THIGH
567A: 30 21    09820          BMI ED
567C: 38       09825 .3       SEC
567D: AD 7A 49 09830          LDA SR10
5680: ED 7E 49 09835          SBC HSR10
5683: 30 18    09840          BMI ED
5685: AD 7A 49 09845 THIGH    LDA SR10     ;UPDATE NEW HIGH SCORE
5688: 8D 7E 49 09850          STA HSR10
568B: AD 7B 49 09855          LDA SR100
568E: 8D 7F 49 09860          STA HSR100
5691: AD 7C 49 09865          LDA SR1000
5694: 8D 80 49 09870          STA HSR1000
5697: AD 7D 49 09875          LDA SR10T
569A: 8D 81 49 09880          STA HSR10T
569D: A9 01    09885 ED       LDA #$01
569F: 8D 82 49 09890          STA DELAY
56A2: A9 00    09895          LDA #$00     ;4 SECOND DELAY
56A4: 85 14    09900          STA $14
```

```
56A6: 85 13     09905           STA $13
56A8: 60        09910           RTS
                09915 *SUBROUTINE TO REPLACE MISSILE BASES
56A9: 8D 2B 85  09920           STA $852B
56AC: 8D 4A 85  09925           STA $854A
56AF: 8D 69 86  09930           STA $8669
56B2: 8D 84 83  09935           STA $8384
56B5: 8D 97 85  09940           STA $8597
56B8: 8D A9 84  09945           STA $84A9
56BB: 8D D3 85  09950           STA $85D3
56BE: A9 64     09955           LDA #$64
56C0: 8D 2B 84  09960           STA $842B
56C3: 8D 4A 84  09965           STA $844A
56C6: 8D 69 85  09970           STA $8569
56C9: 8D 84 82  09975           STA $8284
56CC: 8D 97 84  09980           STA $8497
56CF: 8D A9 83  09985           STA $83A9
56D2: 8D D3 84  09990           STA $84D3
56D5: EA        09995           NOP
56D6: A9 00     10000 MBSUB     LDA #$00
56D8: 8D 86 49  10005           STA MBCOUNT
56DB: A9 63     10010           LDA #$63    ;MISSILES
56DD: 8D 2B 85  10015           STA $852B
56E0: 8D 4A 85  10020           STA $854A
56E3: 8D 69 86  10025           STA $8669
56E6: 8D 84 83  10030           STA $8384
56E9: 8D 97 85  10035           STA $8597
56EC: 8D A9 84  10040           STA $84A9
56EF: 8D D3 85  10045           STA $85D3
56F2: A9 64     10050           LDA #$64
56F4: 8D 2B 84  10055           STA $842B
56F7: 8D 4A 84  10060           STA $844A
56FA: 8D 69 85  10065           STA $8569
56FD: 8D 84 82  10070           STA $8284
5700: 8D 97 84  10075           STA $8497
5703: 8D A9 83  10080           STA $83A9
5706: 8D D3 84  10085           STA $84D3
5709: 60        10090           RTS
                10095 *TEST TARGET HIT SUBROUTINE
570A: AD 8A 49  10100 RTARGET   LDA POS     ;CALC PLAYFIELD POS OF BOMB OR MISSILE
570D: 38        10105           SEC
570E: E9 20     10110           SBC #$20
5710: 18        10115           CLC
5711: 6D 0E 49  10120           ADC FS
5714: 4A        10125           LSR         ;DIVIDE BY 8
5715: 4A        10130           LSR
5716: 4A        10135           LSR
5717: 18        10140           CLC
5718: 6D 0D 49  10145           ADC XS
571B: 8D 84 49  10150           STA BTARGET
571E: CE 84 49  10155           DEC BTARGET ;ALSO TEST VALUE ON EACH SIDE
                10160 *CHECK FOR LASER BASE COLLISION
5721: A9 00     10165           LDA #$00
5723: 8D 85 49  10170           STA STRIKE
5726: A9 3E     10175 LB        LDA #$3E
5728: CD 84 49  10180           CMP BTARGET
572B: F0 07     10185           BEQ .1
572D: A9 3F     10190           LDA #$3F
572F: CD 84 49  10195           CMP BTARGET
5732: D0 0E     10200           BNE .2
5734: A9 00     10205 .1        LDA #$00    ;SHUT OFF LASER BASE 1
```

```
5736: 8D 3E 83 10210          STA $833E
5739: 8D 3F 83 10215          STA $833F
573C: 8D 6E 49 10220          STA GALIVE
573F: 4C 93 57 10225          JMP LBEND
5742: A9 71    10230 .2       LDA #$71
5744: CD 84 49 10235          CMP BTARGET
5747: F0 07    10240          BEQ .3
5749: A9 72    10245          LDA #$72
574B: CD 84 49 10250          CMP BTARGET
574E: D0 0E    10255          BNE .4
5750: A9 00    10260 .3       LDA #$00    ;SHUT OFF LASER BASE 2
5752: 8D 71 86 10265          STA $8671
5755: 8D 72 86 10270          STA $8672
5758: 8D 6F 49 10275          STA GALIVE+1
575B: 4C 93 57 10280          JMP LBEND
575E: A9 9E    10285 .4       LDA #$9E
5760: CD 84 49 10290          CMP BTARGET
5763: F0 07    10295          BEQ .5
5765: A9 9F    10300          LDA #$9F
5767: CD 84 49 10305          CMP BTARGET
576A: D0 0E    10310          BNE .6
576C: A9 00    10315 .5       LDA #$00    ;SHUT OFF LASER BASE 3
576E: 8D 9E 86 10320          STA $869E
5771: 8D 9F 86 10325          STA $869F
5774: 8D 70 49 10330          STA GALIVE+2
5777: 4C 93 57 10335          JMP LBEND
577A: A9 CB    10340 .6       LDA #$CB
577C: CD 84 49 10345          CMP BTARGET
577F: F0 07    10350          BEQ .7
5781: A9 CC    10355          LDA #$CC
5783: CD 84 49 10360          CMP BTARGET
5786: D0 18    10365          BNE MB
5788: A9 00    10370 .7       LDA #$00    ;SHUT OFF LASER BASE 4
578A: 8D CB 86 10375          STA $86CB
578D: 8D CC 86 10380          STA $86CC
5790: 8D 71 49 10385          STA GALIVE+3
5793: 20 9D 55 10390 LBEND    JSR SCORE3
5796: A9 20    10395          LDA #$20    ;TURN ON EXPLOSION SOUND
5798: 8D 90 49 10400          STA SETIME
579B: A9 01    10405          LDA #$01
579D: 8D 85 49 10410          STA STRIKE
               10415 *CHECK AGAINST EACH OF 7 MISSILES
57A0: A0 00    10420 MB       LDY #$00
57A2: B9 AF 48 10425 .1       LDA MBPOSL,Y
57A5: CD 84 49 10430          CMP BTARGET
57A8: F0 08    10435          BEQ .2
57AA: C8       10440          INY
57AB: C0 07    10445          CPY #$07
57AD: D0 F3    10450          BNE .1
57AF: 4C D8 57 10455          JMP MBEND
57B2: B9 AF 48 10460 .2       LDA MBPOSL,Y ;STORE IN 0 PAGE
57B5: 85 F6    10465          STA MBZL
57B7: B9 B6 48 10470          LDA MBPOSH,Y
57BA: 85 F7    10475          STA MBZH
57BC: A0 00    10480          LDY #$00
57BE: A9 00    10485          LDA #$00    ;REMOVE MISSILE BASE
57C0: 91 F6    10490          STA (MBZL),Y
57C2: E6 F7    10495          INC MBZH
57C4: A9 00    10500          LDA #$00
57C6: 91 F6    10505          STA (MBZL),Y
57C8: 20 C1 55 10510          JSR SCORE
```

```
57CB: A9 20    10515          LDA #$20    ;TURN ON EXPLOSION SOUND
57CD: 8D 90 49 10520          STA SETIME
57D0: EE 86 49 10525          INC MBCOUNT
57D3: A9 01    10530          LDA #$01
57D5: 8D 85 49 10535          STA STRIKE
57D8: AD 85 49 10540 MBEND    LDA STRIKE
57DB: C9 01    10545          CMP #$01    ;TARGET REMOVED?
57DD: F0 06    10550          BEQ .1
57DF: EE 84 49 10555          INC BTARGET
57E2: 4C 26 57 10560          JMP LB
57E5: 60       10565 .1       RTS
               10570 *CHANCE SUBROUTINE - 0 TO 4
57E6: AD 0A D2 10575 CHANCE   LDA RANDOM
57E9: 4A       10580          LSR         ;DIVIDE BY 32
57EA: 4A       10585          LSR
57EB: 4A       10590          LSR
57EC: 4A       10595          LSR
57ED: 4A       10600          LSR
57EE: C9 05    10605          CMP #$05
57F0: B0 F4    10610          BGE CHANCE  ;TOO BIG TRY AGAIN
57F2: 8D 87 49 10615          STA LUCK
57F5: 60       10620          RTS
               10625 *DEREZ SUBROUTINE
57F6: A2 00    10630 EXPLODE  LDX #$00
57F8: BD 48 49 10635          LDA TEMPL,X
57FB: 85 F4    10640          STA SHPMOL
57FD: BD 4C 49 10645          LDA TEMPH,X
5800: 85 F5    10650          STA SHPMOH
5802: AD 0A 49 10655          LDA YPM0
5805: 85 F2    10660          STA SHPML
5807: A9 88    10665          LDA /PDATA
5809: 18       10670          CLC
580A: 69 04    10675          ADC #$04
580C: 85 F3    10680          STA SHPMH
580E: A9 48    10685          LDA /SHIP
5810: 85 F1    10690          STA SHPH
5812: A9 4C    10695          LDA #SHIP
5814: 85 F0    10700          STA SHPL
5816: A0 00    10705          LDY #$00    ;COUNTER
5818: A9 00    10710          LDA #$00    ;NEED 0 TO ERASE EACH TIME
581A: 91 F4    10715 .1       STA (SHPMOL),Y ;ERASE OLD SHAPE FIRST
581C: C8       10720          INY
581D: C0 08    10725          CPY #$08
581F: 90 F9    10730          BLT .1
5821: A0 00    10735          LDY #$00    ;START WITH 0TH BYTE IN SHAPE
5823: AD 93 49 10740 .2       LDA EXCOUNT ;FIRST TIME?
5826: D0 05    10745          BNE .22
5828: B1 F0    10750          LDA (SHPL),Y;GET BYTE FROM PROPER SHAPE TABLE
582A: 99 96 49 10755          STA DEREZ,Y ;DO THIS FIRST TIME
582D: AD 0A D2 10760 .22      LDA RANDOM
5830: 0D 0A D2 10765          ORA RANDOM
5833: 39 96 49 10770          AND DEREZ,Y
5836: 99 96 49 10775          STA DEREZ,Y ;TEMP STORE DEGRADED SHAPE
5839: B1 F0    10780          LDA (SHPL),Y
583B: 2D 0A D2 10785          AND RANDOM  ;DEGRADE IMAGE RANDOMILY
583E: 2D 0A D2 10790          AND RANDOM
5841: 2D 0A D2 10795          AND RANDOM
5844: 2D 0A D2 10800          AND RANDOM
5847: 19 96 49 10805          ORA DEREZ,Y ;COMBINE 2 DEGRADED IMAGES SO LESS DEGRADED
584A: 91 F2    10810          STA (SHPML),Y ;PUT IN P/M AREA
584C: C8       10815          INY         ;NEXT BYTE IN SHAPE
```

```
584D: C0 08     10820          CPY #$08     ;DONE?
584F: 90 D2     10825          BLT .2
5851: A5 F2     10830          LDA SHPML    ;TRANSFER NEW P/M POS TO OLD POS
5853: 9D 48 49  10835          STA TEMPL,X
5856: A5 F3     10840          LDA SHPMH
5858: 9D 4C 49  10845          STA TEMPH,X
585B: 60        10850          RTS
                10855 *SOUND SUBROUTINE
585C: AD 8E 49  10860 SOUND    LDA SLTIME   ;CHECK LASER TIMER FLAG
585F: F0 2F     10865          BEQ SOUND2   ;IF 0 SKIP
5861: C9 0F     10870          CMP #$0F     ;TIMER GOES FROM 1 TO 15
5863: D0 0B     10875          BNE .1
5865: A9 00     10880          LDA #$00
5867: 8D 00 D2  10885          STA AUDF1
586A: 8D 01 D2  10890          STA AUDC1
586D: 4C 90 58  10895          JMP SOUND2   ;LEAVE
5870: AD 8F 49  10900 .1       LDA SLTIME1 ;CHECK DELAY TIMER
5873: D0 09     10905          BNE .2       ;IF NOT 0 COUNTDOWN TILL IT IS
5875: AD 8D 49  10910          LDA DELAY1   ;GET NEW DELAY VALUE
5878: 8D 8F 49  10915          STA SLTIME1 ;STORE IT
587B: EE 8E 49  10920          INC SLTIME   ;INCREMENT MAIN TIMER (ALSO OUR FREQ. VALUE
587E: CE 8F 49  10925 .2       DEC SLTIME1 ;COUNTDOWN DELAY TIMER
5881: AD 8E 49  10930          LDA SLTIME   ;OUR FREQ. VALUE
5884: 0A        10935          ASL          ;MULTIPLY BY 16
5885: 0A        10940          ASL
5886: 0A        10945          ASL
5887: 0A        10950          ASL
5888: 8D 00 D2  10955          STA AUDF1    ;NEW TONE VALUE
588B: A9 86     10960          LDA #$86     ;DISTORTION 8 VOLUME 6
588D: 8D 01 D2  10965          STA AUDC1
5890: AD 90 49  10970 SOUND2   LDA SETIME   ;CHECK EXPLOSION TIMER FLAG
5893: F0 0C     10975          BEQ SOUND3   ;IF AT 0 NO SOUND
5895: CE 90 49  10980          DEC SETIME   ;COUNTDOWN
5898: 4A        10985          LSR          ;DIVIDE BY 2 TO GET VOLUME 0-16
5899: 8D 01 D2  10990          STA AUDC1    ;TELL POKEY NEW SOUND VOLUME-
                10992 ;                     ;-UPPER NIBBLE (DISTORTION) IS AT 0
589C: A9 40     10995          LDA #$40     ;TONE
589E: 8D 00 D2  11000          STA AUDF1
58A1: AD 91 49  11005 SOUND3   LDA SEXTIME ;CHECK EXPLOSION TIMER FLAG
58A4: F0 0D     11010          BEQ .1       ;IF AT 0 NO SOUND
58A6: CE 91 49  11015          DEC SEXTIME ;COUNTDOWN
58A9: 4A        11020          LSR          ;DIVIDE BY 4 TO GET VOLUME 0-16
58AA: 4A        11025          LSR
58AB: 8D 07 D2  11030          STA AUDC4    ;TELL POKEY NEW SOUND VOLUME-
                11032 ;                     ;-UPPER NIBBLE (DISTORTION IS AT 0
58AE: A9 40     11035          LDA #$40     ;TONE
58B0: 8D 06 D2  11040          STA AUDF4
58B3: 60        11045 .1       RTS
```

CHAPTER 8

RASTER GRAPHICS & SOUND

Raster graphics is a term we very rarely use in connection with the Atari computer system. It is a term that describes how individual pixels are mapped on a high-resolution screen. The technique is about the only one possible on computers such as the Apple II and the IBM PC. Atari programmers like to use easier and more colorful techniques like character graphics and player-missile animation, but there are certainly a number of valid reasons for animating with raster graphics. The two best reasons are that Graphics mode 8 (ANTIC mode F) screens have the highest resolution (320 x 192 pixels), and that very large shapes can be smoothly animated. The biggest disadvantage is that you can have shapes with three colors at best.

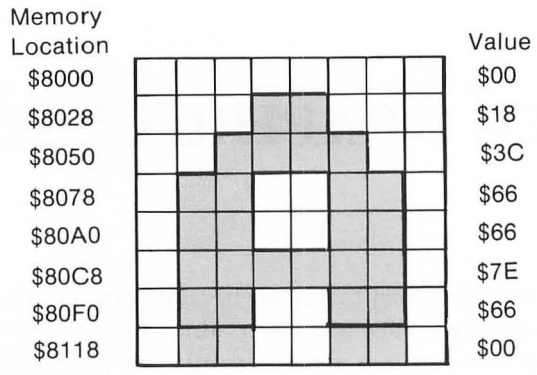
Graphics 8 screens produce color by a method known as artifacting. On computers with a GTIA chip, pixels in even columns appear blue and those in odd columns appear green when the background color register is set to black. Obviously, we could obtain other color combinations by varying the background color register. If you wish to draw a shape entirely in blue, you need only plot the shape's individual pixels in the even columns. Similarly, you will obtain an all-green shape if you plot pixels only in the odd columns. When a blue pixel is next to a green pixel, the pair appears as white.

You get these alternating stripes of color because the Atari sends its color signal as a series of square wave pulses. One complete cycle is called a color clock. When the square wave is high you get blue, and when it is low you get green. Other colors are produced by phase shifting the square waves. These colors have nothing to do with the positions of the actual phosphors on the television tube.

Pixel information is encoded eight pixels per byte. The screen is made up of 192 rows of forty bytes. Forty bytes times eight pixels per byte gives a horizontal resolution of 320 pixels. If you are working in color you are really only talking about half that, or 160 pixel pairs horizontally.

Plotting pixel data is quite anolagous to plotting character data to the screen. In fact, if we took the character data for the letter "A" and plotted it on the screen by calculating the byte address for a particular column in the first eight rows, the character would appear as expected. Of course, it is a lot of trouble to just plot character data on a Graphics 8 screen, but it only illustrates the technique.

A Graphics 8 screen fortunately can be mapped sequentially in memory if you are clever. The problem is that a display list cannot address screen memory that crosses a 4K boundary. An additional LMS instruction is needed on the other side of the boundary. You could fit 102 lines in the lower portion, but that would leave a 16-byte



gap between the two sections. Instead, I decided to place 102 lines in the following example in the top 4K of screen memory and butt the first ninety lines against it in the lower 4K section. This leaves 496 bytes of free memory at the beginning and ample room to place the display list. The display list begins at $6000, and screen memory at $61F0. The 0th or top line starts at that address and the first line begins forty bytes later at $6218. Each of the 192 lines is offset in memory by an additional 40 bytes.

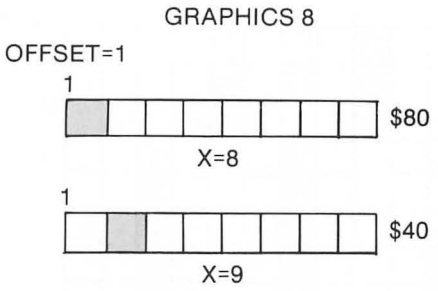
To plot a byte at a particular X,Y coordinate on the screen requires you to calculate a memory address based on the starting address of the row and the offset into the row. The formula is:

MEMORY ADDRESS = SCREEN MEMORY +(Y*40)+(X/8)

It isn't a difficult calculation, except that in Machine language, multiplication and division other than by multiples of eight require an enormous number of steps. If you only had to do it once, it would be alright. Unfortunately, you need to perform the calculation at least once for each row of the shape. If you were trying to do a *Galaxian*-type game where you had several dozen shapes, you would never have enough time to move and draw them all and achieve a fast enough animation frame rate. A better method is to look up the starting address of the row from a table and just calculate the horizontal offset based on the shape's Y coordinate.

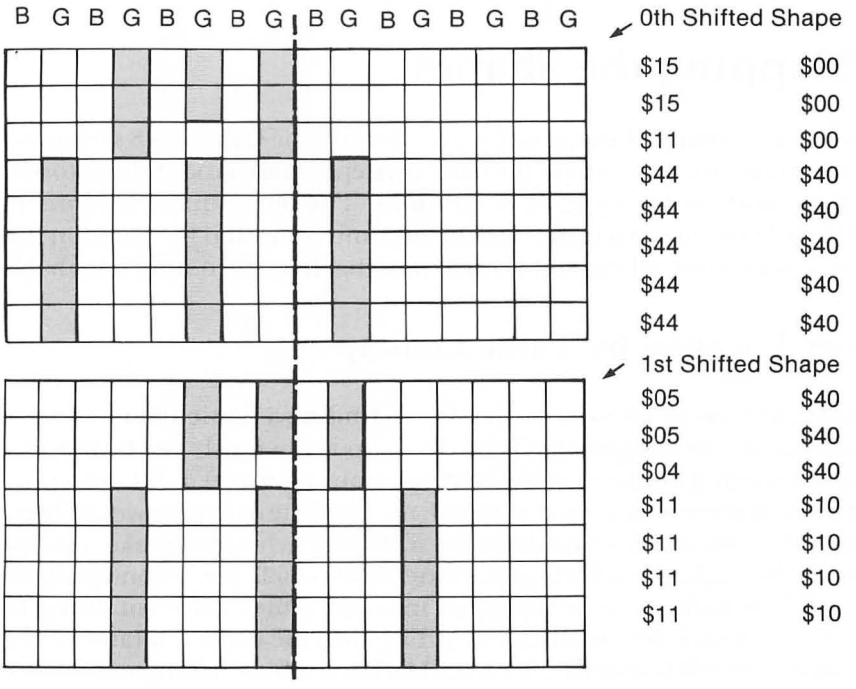
Plotting a pixel on the screen at a particular X,Y coordinate, based on its row and horizontal offset, will work only if the Y coordinate was a multiple of eight. The pixel would be physically at the left end of the byte and would have a value of $F0. If you want to plot a pixel one unit further to the right, you need a byte with an entirely different pixel pattern. You can't physically move the byte just anywhere, like a "tad" to the right. The value of that byte is $80. Now, you begin to realize that you need eight different bytes just to cover all of the possible X coordinates. Since this is also true of larger shapes, we have a difficult problem.

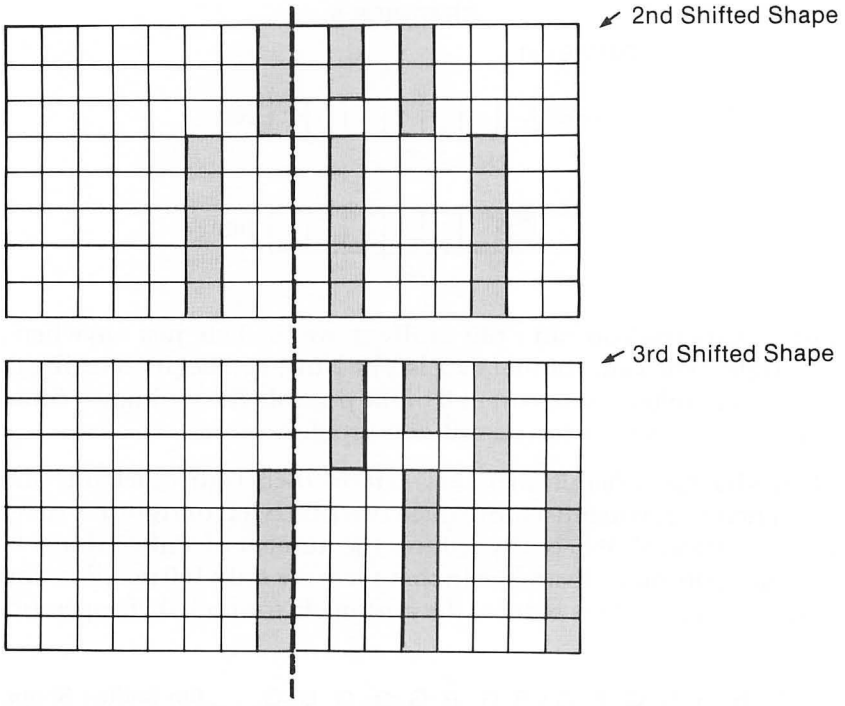
Plotting a pixel on the screen at a particular X,Y coordinate, based on its row and horizontal offset, will work only if the Y coordinate was a multiple of eight. The pixel would be physically at the left end of the byte and would have a value of $80. If you want to plot a pixel one unit to the right, you need a byte with an entirely



different pixel pattern. You can't physically move the byte just anywhere, like a "tad" to the right. The value of that byte is $40. Now, you begin to realize that you need eight different bytes just to cover all of the possible X coordinates. Since this is also true of larger shapes, we have a difficult problem.

Color shapes have another problem. If you move them right or left one pixel, they shift colors. Therefore, you must move them two pixels left or right at a time. While this sounds complicated, it actually reduces the number of shifted shape tables to four. It has one additional advantage in that there are only 160 possible horizontal positions instead of 320. This reduces the arithmetic to single-byte operations.





Bit Mapping the Shapes

Drawing a bit-mapped shape table anywhere on the Graphics 8 screen is a simple procedure, once you understand the basic concept. The shape table is stored sequentially in memory, either by rows or columns. The technique, therefore, is to load each of these bytes, one at a time, into the Accumulator, find the position in memory for the screen location where you want to plot that byte, then store it in that location.

Memory Location by Table Lookup

The difficulty, as we showed earlier, lies in finding a particular memory location, given an X,Y screen coordinate. Table look-up is obviously the fastest method for finding the starting address for the first position (leftmost) or 0th offset for each of the 192 lines. If the screen started at $61F0, the first line or line #0 would begin at this address, and the second line would begin at $6218. Each address takes two bytes. The first part is the high byte which in the latter case is $62. The second part, $18, is the low byte. These values can be separated into two tables, one containing the lower order address of each line (call it YVERTL), and the other containing the higher order address of each line (call it YVERTH). Each table is 192 bytes long (0 -191). In order that these tables not become specific to a particular screen address, the values are merely offset values from a zero starting address. The GETADR subroutine adds the high byte of the starting screen address to obtain a specific memory address. Our only constraint is that the screen start on a page boundary.

You can access any element in either table by absolute indexed addressing. The effective address of the operand is computed by adding the contents of the Y register to the address of the instruction. The format is:

EFFECTIVE ADDRESS = ABSOLUTE ADDRESS + Y REGISTER

If our YVERTL table were stored at $4000 and we wanted to find the starting address of line 1 (remember lines are numbered 0-191), we would index into the table one position and load that value into the Accumulator.

4000:F0 18 40 68 90 B8 .....YVERTL TABLE

So LDA YVERTL,Y, where the Y register =$01, will fetch the value $18 from memory location $4000 + $01 = $4001, and place it in the Accumulator.

Similarly, if YVERTH were stored in the next page following our first table, then:

4100:01 02 02 02 02 02 .....YVERTH TABLE

If the Y register = $01, then a LDA YVERTL,Y will take the value $02 stored in memory location $4100+ $01 = $4101, and place it in the Accumulator.

Storing the Shape in Screen Memory

Eventually we will want to store the first byte from the shape table into a memory location. This can be done efficiently if the two-byte address is stored sequentially in zero page. Let's store the low-byte half of the address, HIRESL, at location $F2, and the high-byte half, HIRESH, at location $F3 in zero page:

```
LDY #$01      ;Y REGISTER CONTAINS LINE #
LDA YVERTH,Y ;LOOKUP HIGH BYTE OF START
              ;OF ROW IN MEMORY
STA HIRESH    ;STORE IN ZERO PAGE OF MEMORY
LDA YVERTL,Y ;LOOKUP LOW BYE OF ROW IN MEMORY
STA HIRESL    ;STORE IN ZERO PAGE OF MEMORY
```

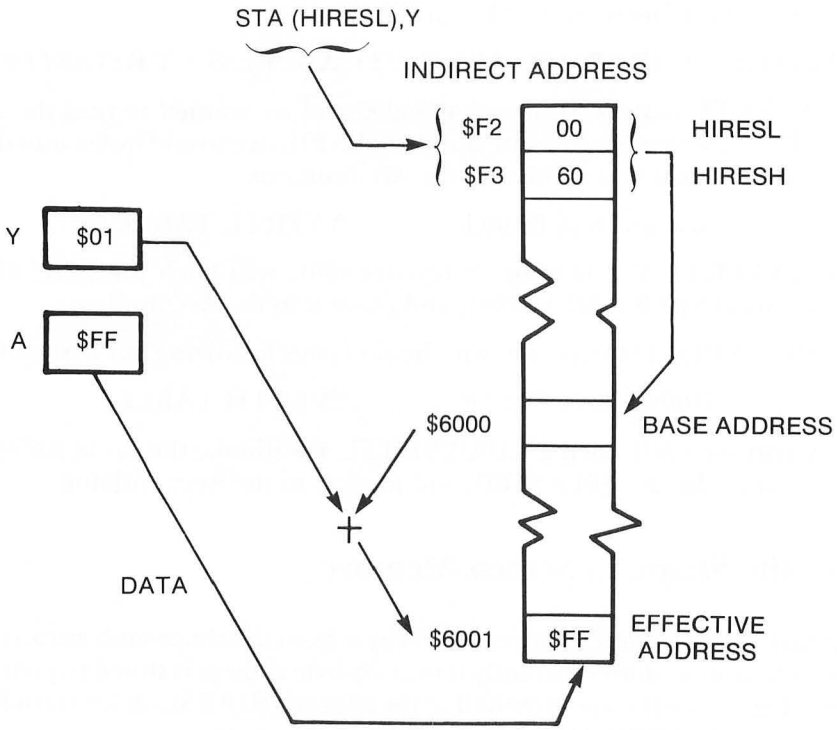
If the computer finds a $00 in location $F2 (HIRESL) and a $60 in location $F3 (HIRESH), then the base address is $6000. The Accumulator stores a value into memory location $60000+$01, or lacation #6001, as shown on the following page.

The final addressing mode that we must consider is Indexed Indirect Addressing. The format is:

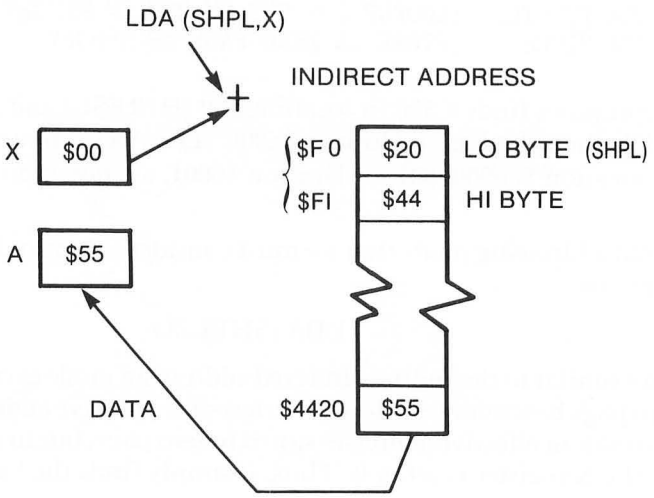
LDA (SHPL,X)

It is very similar to the Indirect Indexed addressing mode except the index is added to the zero page base address before it retrieves the effective address. Its primary use is to index a table of effective addresses stored in zero page, but in the form we are going to use it, the X register is set to 0. Thus, it simply finds the base address.

INDIRECT INDEXED ADDRESSING



INDEXED INDIRECT ADDRESSING



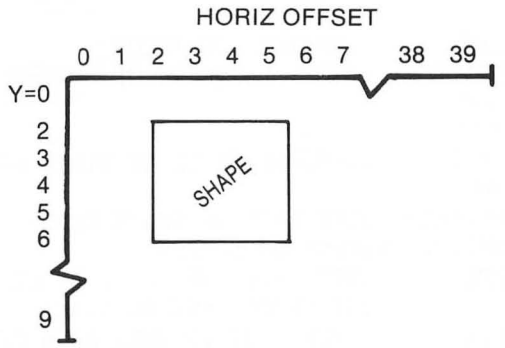
We must use this second form of indirect addressing because there is a shortage of registers in the 6502 microprocessor. We are already using the Y register in the store operation, and there isn't an indirect indexed addressing mode of the form LDA(SHPL),X. Thus, we must go to the alternative addressing mode LDA(SHPL,X).

What this all boils down to is that we want to load a byte from a shape table into the Accumulator and store it on the screen with the following instructions:

```
LDA (SHPL,X)    ;LOAD BYTE FROM SHAPE TABLE
STA (HIRESL),Y  ;STORE BYTE ON HI-RES SCREEN
```

We can index into the shape table by incrementing the low byte SHPL by one each time, then store that byte into the next screen position on a particular line by incrementing the Y register. This zero page method is faster than performing the equivalent code with absolute index addressing, because two-byte addresses can be handled with fewer instructions, fewer machine cycles, and less memory space.

Obviously, a generalized subroutine must be developed to find the screen memory address (HIRESL and HIRESH), given a line number and a horizontal displacement. We will call this subroutine GETADR, short for Get Address.



Each time a row of shape-table bytes is transferred to succesive memory locations in screen memory, the program will call the subroutine GETADR. The line's starting memory address is then offset by the horizontal location of the shape on the screen. Our table of line addresses is only an offset, so it will need to add the actual starting address of the screen.

Memory address = Line # starting address + horizontal offset

```
GETADR   LDA YVERTL,Y ;LOOKUP LOW BYTE ADDRESS OF LINE
         CLC
         ADC HORIZ    ;ADD HORIZ. OFFSET
         STA HIRESL   ;STORE LOW BYTE OF SCREEN ADDRESS
         LDA YVERTH,Y ;LOOKUP HIGH BYTE ADDRESS OF LINE
         ADC /SCREEN  ;ADD HIGH BYTE OF SCREEN ADDRESS
         STA HIRESH   ;STORE HIGH BYTE SCREEN ADDRESS
         RTS
```

where the Y register has a vertical screen value (0-191).

If you are designing an arcade game, you will probably have several different shapes on the screen at one time. Keeping track of each shape's variables, which are inputted into a generalized drawing routine, is generally easier if a set-up routine is incorporated into your program. This assures that you haven't forgotten to initialize anything before entering the drawing routine. Only a few variables need to be defined in the set-up routine: the location of the shape table; the horizontal displacement on the screen; and the width and depth of the shape.

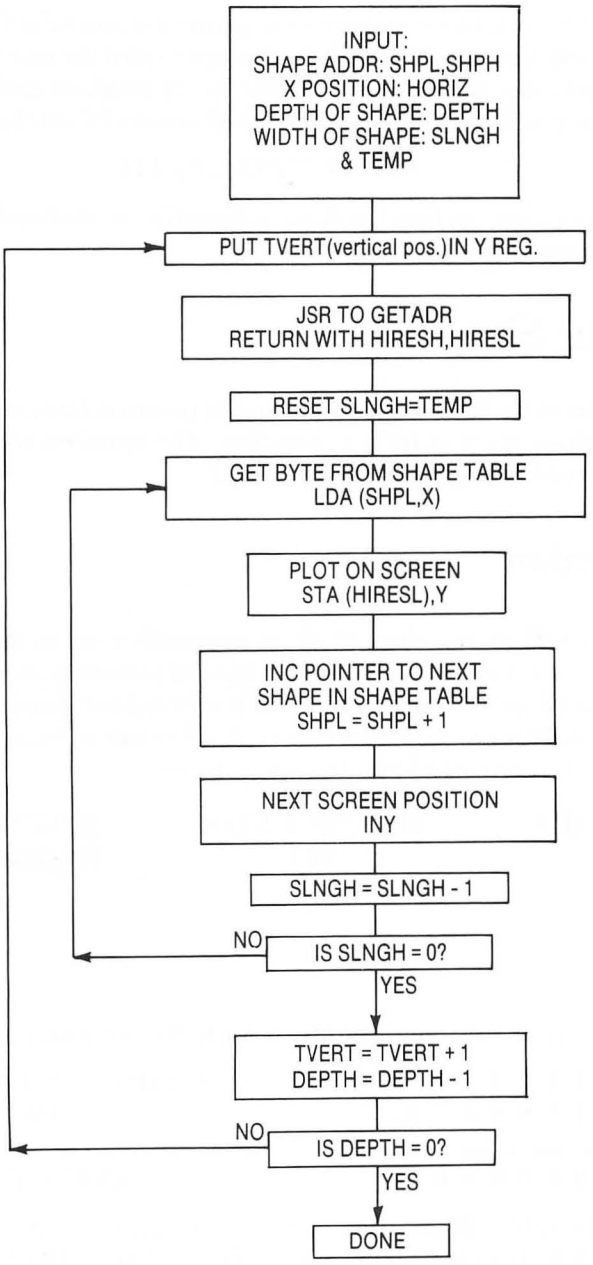
The drawing routine becomes more efficient the fewer times it accesses the GETADR subroutine. Therefore, it is much faster to load and store on the same screen line until the end of the shape's width is reached. Drawing our balloon a byte at a time across its width will only require calling GETADR 31 times. But if we plotted down instead, GETADR would be called for each byte, or 279 times, an unnecessary waste of time.

As we load and store across a particular screen line, we decrement SLNGH, the ship's width, until SLNGH equals zero. When we are finished with a row, we increment TVERT to the next screen line down and decrement the DEPTH. When DEPTH reaches zero, we have plotted all rows of the shape and we are finished.

```
DRAW    LDY VERT       ;VERTICAL POSITION
        JSR GETADR     ;FIND BEGINNING OF SCREEN ADDRESS ROW
        LDX #$00
        LDA TEMP
        STA SLNGH      ;RESTORE VALUE OF WIDTH FOR NEXT ROW
        LDY #$00
DRAW2   LDA (SHPL,X)   ;GET BYTE OF SHAPE TABLE
        STA (HIRESL),Y;PLOT ON SCREEN
        INC SHPL       ;NEXT BYTE OF SHAPE TABLE
        BNE .1         ;IF CROSS PAGE BOUNDARY?
        INC SHPH       ;INCREMENT TO NEXT PAGE OF SHAPE
.1      INY            ;NEXT POSITION ON SCREEN
        DEC SLNGH      ;DECREMENT WIDTH
        BNE DRAW2      ;FINISHED WITH ROW YET?
        INC VERT       ;IF SO, INCREMENT TO NEXT LINE
        DEC DEPTH      ;DECREMENT DEPTH
        BNE DRAW       ;FINISHED ALL ROWS?
        RTS            ;YES, END
```

0	1	2
3	4	5
6	7	8
9	...	

Map of elements in shape table as they appear on the screen


INPUT:
SHAPE ADDR: SHPL,SHPH
X POSITION: HORIZ
DEPTH OF SHAPE: DEPTH
WIDTH OF SHAPE: SLNGH
& TEMP
PUT TVERT(vertical pos.)IN Y REG.
JSR TO GETADR
RETURN WITH HIRESH,HIRESL
RESET SLNGH=TEMP
GET BYTE FROM SHAPE TABLE
LDA (SHPL,X)
PLOT ON SCREEN
STA (HIRESL),Y
INC POINTER TO NEXT
SHAPE IN SHAPE TABLE
SHPL = SHPL + 1
NEXT SCREEN POSITION
INY
SLNGH = SLNGH - 1
NO
IS SLNGH = 0?
YES
TVERT = TVERT + 1
DEPTH = DEPTH - 1
NO
IS DEPTH = 0?
YES
DONE

Although the first row of the shape can be plotted at any VERT (0 -191) position, if VERT began at 190, the computer would attempt to plot the third line at VERT=192. Indexing into the table that far would most likely produce garbage, as you would index beyond the end of the table. You should always be careful that:

$$TVERT \leqslant 192-DEPTH$$

A simple test somewhere before the draw subroutine would suffice, but it might be incorporated into your joystick read routine.

XDrawing Shapes

Objects that move on the screen are shifted in position by erasing the object's first position before drawing it at its new position. The simplest method is to draw the shape by Exclusive-ORing it before shifting it.

EOR Instruction

The Exclusive-OR instruction, EOR, is primarily used to determine which bits differ between two operands, but it can also be used to complement selected Accumulator bits. The way it works is elementary. If neither of two particular memory bits is set or their values are zero, the result is zero. If either one is set, then the result is one. But if both are set, they cancel and the result is zero.

	MEMORY BIT	ACCUMULATOR BIT	RESULT BIT IN ACCUMULATOR
	0	0	0
EOR	0	1	1
	1	0	1
	1	1	0

If we take a byte on the screen and EOR it with the same byte

```
  0 1 1 0 0 1 1 0     SHAPE ON SCREEN
  0 1 1 0 0 1 1 0     SHAPE
 -----------------
  0 0 0 0 0 0 0 0     RESULT
```

From the shape table, the result is zero or a screen erase. A similar effect would occur if a blank screen were EORed with a shape, then EORed again.

```
                0 0 0 0 0 0 0 0     BLANK SCREEN
EOR             0 1 1 0 0 1 1 0     WITH SHAPE
               -----------------
                0 1 1 0 0 1 1 0     RESULT IS SHAPE ON SCREEN
EOR             0 1 1 0 0 1 1 0
               -----------------
                0 0 0 0 0 0 0 0     RESULT IS BLANK SCREEN
```

It doesn't damage the background if a shape is EORed on the screen, and then off again. However it does distort the shape slightly.

```
          0 0 0 0 0 0 0 1      BACKGROUND
EOR       0 0 1 0 1 1 0 0      WITH SHAPE
          -----------------
          0 0 1 0 1 1 0 1      RESULT ON SCREEN (SHAPE
                                 DISTORTED LAST BIT)
EOR       0 0 1 0 1 1 0 0      WITH SHAPE
          -----------------
          0 0 0 0 0 0 0 1      GET BACKGROUND BACK
```

In the above example, an extra pixel in the shape's last bit position distorts the shape drawn on the screen. In the example below, the fourth bit position becomes a hole in the shape.

```
          0 0 0 1 0 0 0 0      BACKGROUND
EOR       0 1 0 1 1 0 0 0      WITH SHAPE
          -----------------
          0 1 0 0 1 0 0 0      RESULT ON SCREEN
                 \_____ hole here
EOR       0 1 0 1 1 0 0 0      WITH SHAPE
          -----------------
          0 0 0 1 0 0 0 0
```

There are some techniques to avoid distorting the shape when the background is likely to interfere during the drawing process. This involves a combination of EORing and ORing the screen with the background stored in an alternate screen memory. An alternate method is to store the screen memory bytes in a temporary table equal in size to your shape, while you draw your shape. When erasing, you replace the shape with the background stored in your temporary table.

OR Instruction

The OR memory with Accumulator (ORA) instruction differs from the EOR instruction in that if both memory and Accumulator bits are on, then the result is one, or on.

	MEMORY BIT	ACCUMULATOR BIT	RESULT BIT IN ACCUMULATOR
	0	0	0
ORA	0	1	1
	1	0	1
	1	1	1

If the background were as follows, and you ORed it with the shape, the shape remains correct.

```
          0 0 1 0 1 0 1 0      BACKGROUND
ORA       0 1 1 1 1 0 0 0      WITH SHAPE
          -----------------
          0 1 1 1 1 0 1 0      GET SHAPE + BACKGROUND
                               WITH NO HOLE IN SHAPE
```

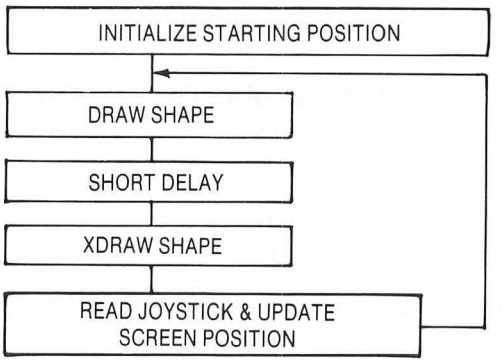
Unfortunately, if you EOR this result with the shape again, the background is flawed.

```
          0 1 1 1 1 0 1 0      SHAPE + BACKGROUND
EOR       0 1 1 1 1 0 0 0      WITH SHAPE
          -----------------
=         0 0 0 0 0 0 1 0      FLAWED BACKGROUND
```

We can incorporate the Exclusive-OR instruction in our XDRAW routine. If we EOR the shape we had previously on the screen, nothing remains.

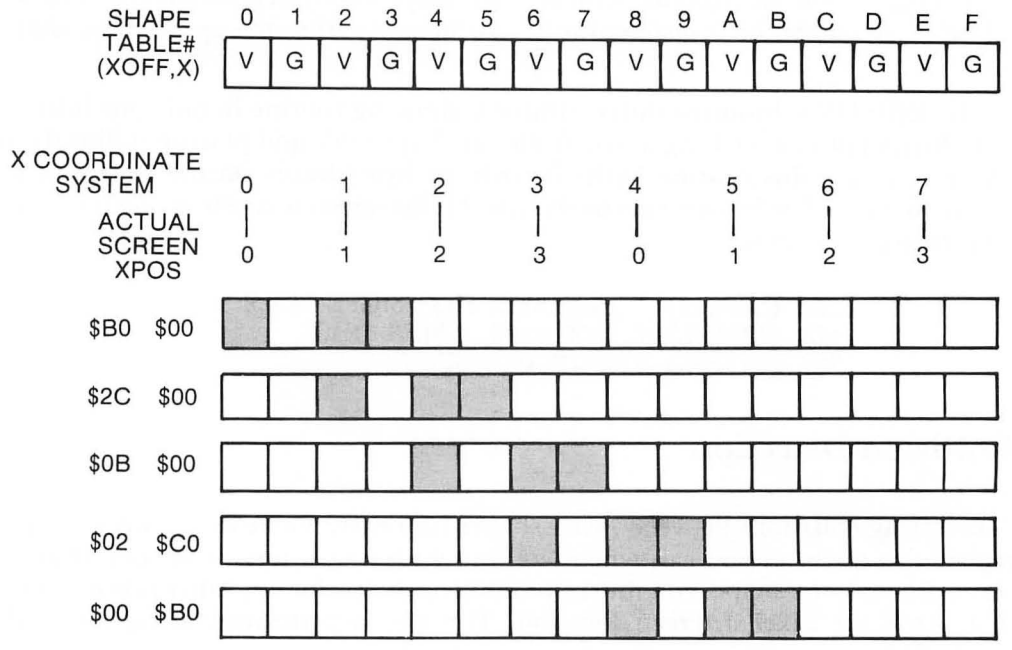
```
00010 XDRAW    LDY TVERT       ;VERTICAL POSITION
00020          JSR GETADR
00030          LDA TEMP
00040          STA SLNGH       ;RESTORE VALUE OF WIDTH FOR NEXT ROW
00050          LDX #$00
00060 XDRAW2   LDA (SHPL,X)    ;GET BYTE FROM SHAPE TABLE
00070          EOR (HIRESL),Y  ;EOR WITH BYTE ALREADY ON SCREEN
00080          STA (HIRESL),Y  ;DRAW ON SCREEN
00090          INC SHPL        ;NEXT BYTE OF SHAPE TABLE
00100          INY
00110          DEC SLNGH       ;DECREMENT WIDTH
00120          BNE DRAW2       ;FINISHED WITH ROW?
00130          INC TVERT       ;IF SO, INCREMENT TO NEXT LINE
00140          DEC DEPTH       ;DECREMENT DEPTH
00150          BNE DRAW        ;FINISHED ALL ROWS?
00160          RTS             ;YES, END ROUTINE
```

Now that we know how to DRAW and XDRAW a bit-mapped shape anywhere on a Graphics mode 8 screen, the principle for animating them is simple. A shape is erased from the screen, its new position is calculated, then it is redrawn at its new position. The procedure is outlined below.



A delay has been inserted between the DRAW and the XDRAW to allow the object to be on the screen longer than it is off. Without the delay, the object is erased immediately after it is drawn. This does not give the shape's image sufficient time to remain onscreen during one animation frame. The result is a badly-flickering image. A small delay can be inserted by checking the internal clock at $14 for so many jiffies. Experiments show that 3/60 seconds is a good value.

Whenever a shape is moved horizontally, the bit pattern within a screen memory byte shifts and sometimes even intrudes into the adjacent byte. To avoid color shifts due to the odd-even column artifacting, shapes must be moved horizontally two columns at a time. If we consider the four-pixel-wide shape in the diagram below and move it right two bits at a time, it retains the same shape and color pattern, but the value in its shape table changes. If we shift the shape far enough, some of the bits run into the next byte. By the time we have shifted it the fourth time, the pattern repeats itself as if it were in the same starting position but one byte over. So if we are going to be able to move a shape anywhere on the screen, we will need four shifted shape tables each one byte wider than the original shape. And since we need to move the shape horizontally two pixels or color clocks at a time, it would be easier to work with a coordinate system that goes from 0-159 instead of 0-319.



It would be nice if there were a relationship between the horizontal position (X) and the shape #. The mathematical relationship is as follows:

TEMP = INT (X/4)
SHAPE# = X–TEMP*4

Actually, it is a lot faster to look the value up in a table called XOFF. This table has the shape table # for each possible X position. You can retrieve the shape table number by indirectly indexing into the table with the X position in the Y-register. We use another small table SHPLO to store the low byte starting positions of each of the shape tables, and another called SHPHI if the combined length of the four shapes crosses a page boundary. The code to set up the pointers to the proper shape is as follows:

```
LDY X        ;HORIZONTAL POSITION (0-159)
LDX XOFF,Y   ;INDEX TO FIND SHAPE #
LDA SHPLO,X ;INDEX TO GET LOW BYTE OF SHAPE TABLE
STA SHPL     ;STORE LOW BYTE IN ZERO PAGE
LDA SHPHI,X ;GET HIGH BYTE OF SHAPE TABLE
STA SHPH     ;STORE HIGH BYTE IN ZERO PAGE
```

The drawing routine is exactly the one described earlier. Once the pointers to the proper shape table are inputted with both the shape's vertical position and horizontal offset, bytes can be transferred to screen memory from the appropriate shape table.

The XDRAW subroutine differs from our drawing routine in only one instruction. Instead of just fetching a byte from our shape table and placing it directly in screen memory, this routine EORs it with the byte already on the screen before storing it there. The bits are effectively erased if the screen image byte and the shape table byte are a match.

```
LDA (SHPL,X)    ;GET BYTE FROM SHAPE TABLE
EOR (HIRESL),Y ;EOR WITH SCREEN IMAGE
STA (HIRESL),Y ;PLOT ON SCREEN
```

Collision Detection

Detecting collisions between raster shapes isn't easy. There aren't any collision registers to query as you can when working with player-missile shapes. Instead, when drawing the shape, you must simultaneously test for any other pixels within that byte's (or pixel's) screen location. The test is performed using the AND instruction.

The AND Instruction

The truth table for the AND instruction is as follows:

ACC.	MEMORY	RESULT
0	0	0
0	1	0
1	0	0
1	1	1

Both Accumulator and memory must be on (set) for the result to be on (set).

If we take a screen memory location that has an object in it and AND it with a byte from our shape table, any duplication in any bit location where something is already on the screen will give a non-zero result.

```
        0 1 1 1 1 0 0 0     Background
AND     0 0 0 1 1 1 1 1     Shape
        -----------
        0 0 0 1 1 0 0 0     Result $18 >Zero
```

Drawing While Testing for Collision

Usually, in any game, if a collision is detected, the object is to be removed. Your first instinct is to stop drawing the object since it is to be removed anyway. But if you are Exclusive-ORing (EORing) the screen and you stop in the middle of your shape, you are going to leave a mess. It is much better to set a collision flag, finish drawing the shape, then remove the object later by completely EORing the shape off the screen.

```
        LDA (SHPL,X)    ;GET BYTE FROM SHAPE TABLE
        AND (HIRESL),Y  ;AND WITH SCREEN IMAGE
        BEQ DRAW        ;BRANCH ON NO COLLISION
        LDA #$01        ;SET COLLISION FLAG
        STA ESET
DRAW    LDA (SHPL,X)    ;GET BYTE FROM SHAPE TABLE
        EOR (HIRESL),Y  ;EOR WITH SCREEN IMAGE
        STA (HIRESL),Y  ;PLOT ON SCREEN
```

Collision Detection — A Special Case

Any two objects of byte size or larger should have no problem with collision detection, especially if you are working with solid white objects. But there is a specific case involving artifacting in which collision detection would not work. Let us assume that we have a blue spaceship and a green alien that appear to collide. If we examine their bit patterns, you will notice that they never coincide.

```
          B G B G B G B G B G B G
          ---------------------
          0 0 1 0 1 0 1 0 1 0 1 0   SHIP
AND       0 0 0 1 0 1 0 1 0 1 0 0   ALIEN
          ---------------------
          0 0 0 0 0 0 0 0 0 0 0 0   RESULT 0
```

The solution is to test the ship against screen memory with what is caled a "mask" of the ship's shape, as if the ship were solid white. We take this mask of the ship, which has both blue and green pixels lit, and AND it against the alien occupying the same screen locations. A collision will be detected in this case. We set a flag, and then take the appropriate byte from the blue ship's shape table and EOR it against the screen.

Blimp Example

A good raster graphics example would be one that would be difficult or impossible to do with either animated character graphics or player-missile graphics. The large elongated blimp shape in this example is eight bytes wide (nine if you count the byte needed for the offset shapes), and thirty-one scan lines deep. By artifacting, we are able to produce a shape of three different colors: blue, green, and white. The shape is outlined below.

The blimp is joystick-controlled and therefore free to move anywhere on the screen. You have to be very careful that you don't try to plot the shape beyond the screen boundaries. While plotting bytes beyond the right edge would produce a wraparound effect at the left edge one scan line lower, plotting beyond scan line 191 could create severe problems by wiping out some portion of memory. If we exceed the bounds of our YVERTL, YVERTH tables, unknown pointers to our plotting position in screen memory would be placed in zero page. In this case, the vertical position cannot exceed 192-31 = 161. All of these tests are incorporated in the joystick subroutine.

Flickering Problem with Large Raster Shapes

The first time we attempted the example, we placed the raster drawing code outside of VBlank and the music routine in VBlank. Unfortunately, the rastered image flickered badly because the image, after remaining on the screen for several frames, has to be erased at some point before it is redrawn at its new position. Since this takes place when the electron beam is on the screen, there is a slight gap, possibly as long as a frame, before the redrawn image is in place. This never seems to be a problem on other computers like the Apple II, but then they use interlacing techniques to produce their television images while the Atari does not.

We then felt that the animation should become flicker-free if we moved the raster drawing routines inside VBlank. This way the rastered image could be erased and

redrawn while the electron beam was mostly off-screen. We arranged the code so that the shape is drawn initially, remains stationary on the screen for three television frames, then is erased, moved, and redrawn on the fourth frame. A timer called TDELAY is set to zero after each erasure, and increments with each frame. A test will cause a branch past the erase-move-redraw code when TDELAY is not equal to three. When it is equal, it will erase the shape, read the joystick, calculate its new position, then redraw it in that position.

We feared that the code might be too long to fit within one Deferred VBlank cycle because the shape was nine bytes by thirty-one scan lines. Having never encountered the problem before, I became confused with the buggy results. The code was arranged differently within the Vblank at the time. The sound routine was last, and I was drawing the raster shape on each frame. The routine invariably drew the shape then hung the first time it was run after assembly, but would actually execute the code after a system reset. A more serious problem was that six complete scan lines directly beneath the shape were garbaged. The routines worked when the code was outside VBlank.

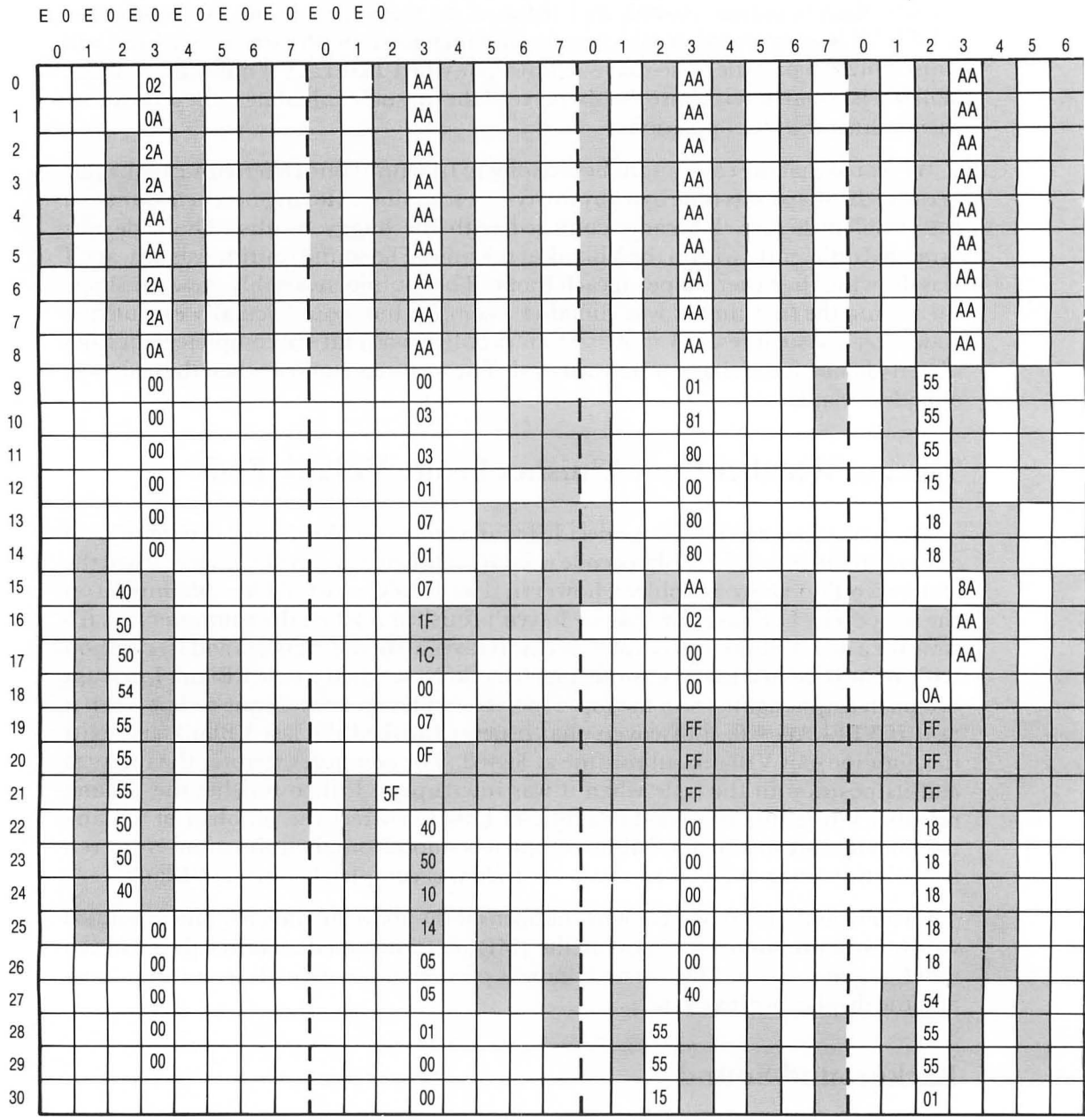
Testing Whether Code Finishes Before VBlank Ends

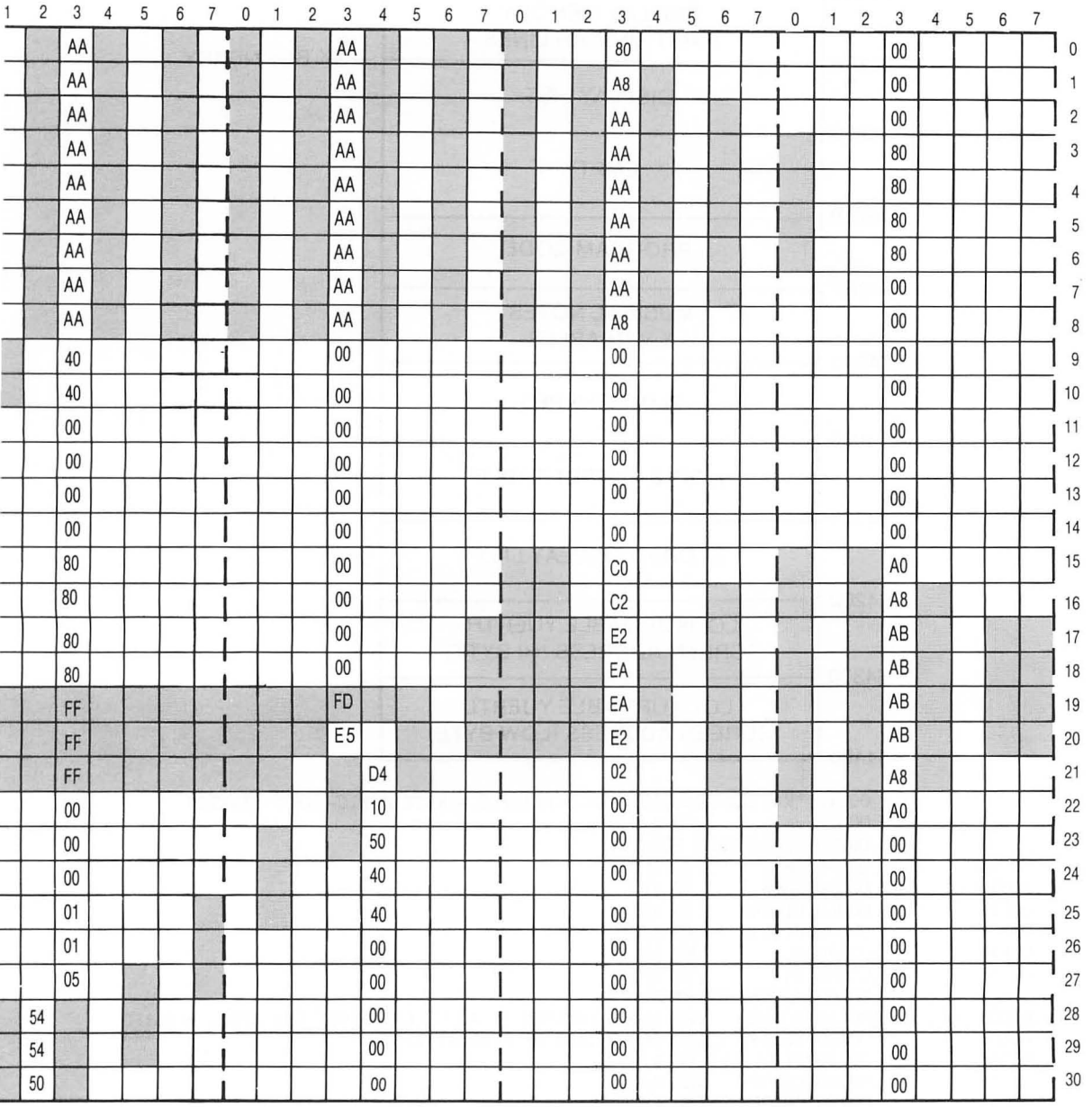
After many wasted hours, we decided that a test would be needed to determine if and when the end of the VBlank code was ever reached. Obviously, if we reached the end we wouldn't have a problem; however, if we didn't, we would have to finish it on the next cycle. Let's assume that we haven't finished it when the computer says it's time for a new VBlank Interrupt to occur. It saves all of the registers and its position within the code just like it was outside VBlank. Now when a new VBlank Interrupt occurs, it begins again from the top. Fine, it executes the sound routine but when it tests if VBFLAG = 0, it discovers that it never finished the last VBlank and exits through the exit VBlank subroutine at $E462. The computer restores the registers and its position in the code when it was interrupted. It then finishes the VBlank routine. While this is a good example of how to correct the problem of VBlank routines that are too long, it fails to completely smooth out the animation. However, it is slightly better than when the raster code was completely outside VBlank.

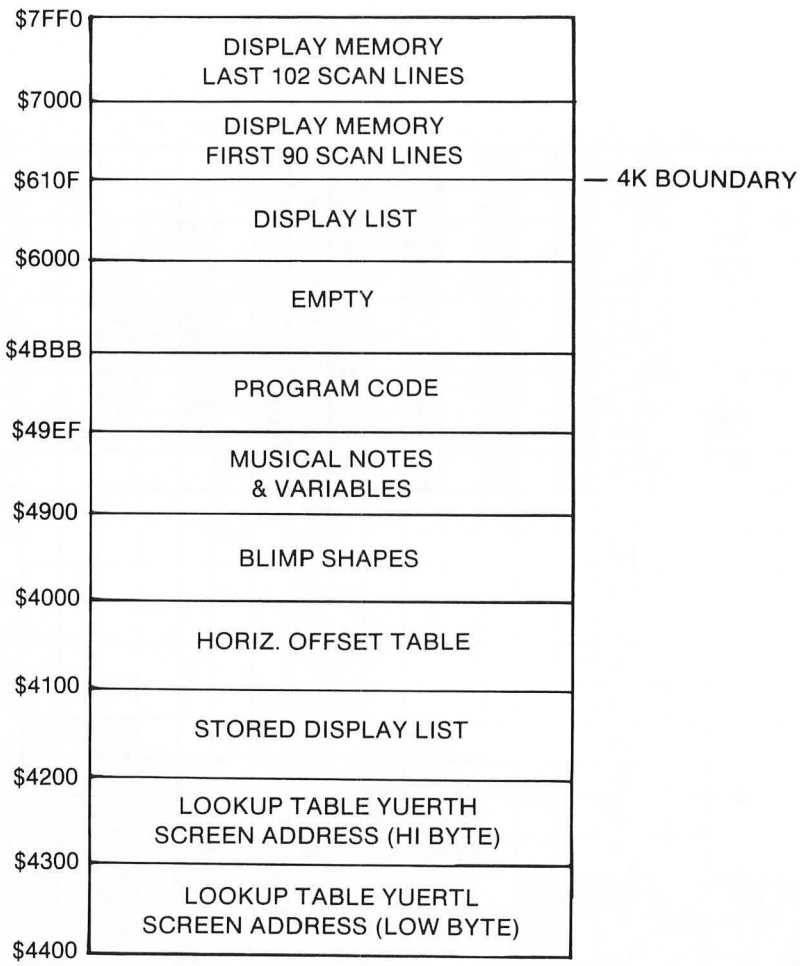
If you would like to observe what happens if the above method isn't incorporated within your program, try removing the JMP $E462 statement. The result is a screen that has gone wacko. The rastered shape is plotted in pieces on different scan lines, and the display begins to roll.

Background Sound

The background sound throughout our raster example is a familiar tune. The sound routine, which is explained in the next section, reads the individual notes and their length from a table. It runs in VBlank because the length of the notes uses the system timers.


E 0 E 0 E 0 E 0 E 0 E 0 E 0 E 0 E 0
0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6
0 02 AA AA AA AA
1 0A AA AA AA AA
2 2A AA AA AA AA
3 2A AA AA AA AA
4 AA AA AA AA
5 AA AA AA AA
6 2A AA AA AA AA
7 2A AA AA AA AA
8 0A AA AA AA AA
9 00 00 01 55
10 00 03 81 55
11 00 03 80 55
12 00 01 00 15
13 00 07 80 18
14 00 01 80 18
15 40 07 AA 8A
16 50 1F 02 AA
17 50 1C 00 AA
18 54 00 00 0A
19 55 07 FF FF
20 55 0F FF FF
21 55 5F FF FF
22 50 40 00 18
23 50 50 00 18
24 40 10 00 18
25 00 14 00 18
26 00 05 00 18
27 00 05 40 54
28 00 01 55 55
29 00 00 55 55
30 00 15 01

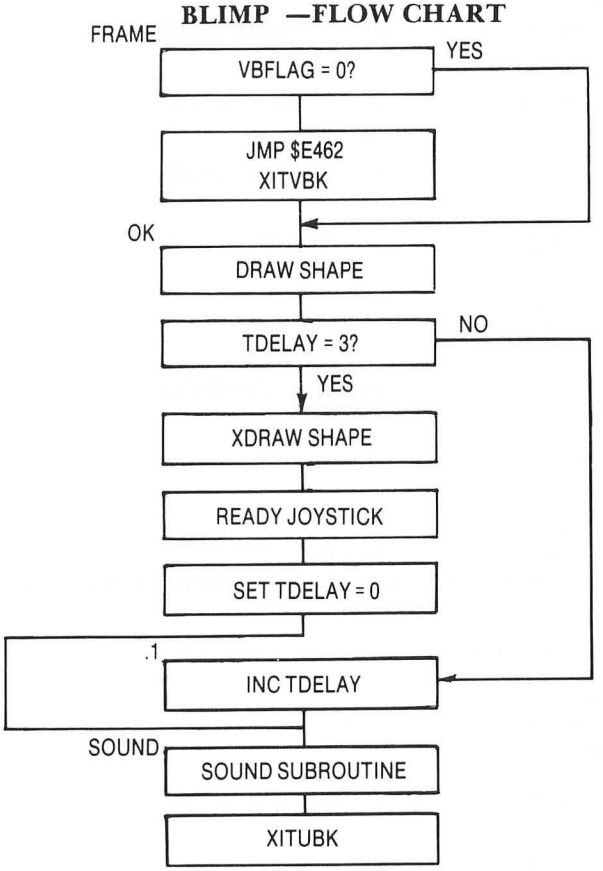

1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
AA AA AA AA AA AA AA AA AA 40 40 00 00 00 00 80 80 80 FF FF FF 00 00 00 01 01 05 54 54 50
AA AA AA AA AA AA AA AA AA 00 00 00 00 00 00 00 00 00 FD E5 D4 10 50 40 40 00 00 00 00 00
80 A8 AA AA AA AA AA AA A8 00 00 00 00 00 00 C0 C2 E2 EA EA E2 02 00 00 00 00 00 00 00 00
00 00 00 80 80 80 80 80 80 00 00 00 00 00 00 A0 A8 AB AB AB AB A8 A0 00 00 00 00 00 00 00
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30




```
                00010 *RASTER GRAPHICS -GR.8 EXAMPLE - WITH MUSIC- JEFF STANTON
                00020          .OR $4000
                00030 *ZERO PAGE EQUATES
00F0:           00040 SHPL     .EQ $F0
00F1:           00050 SHPH     .EQ $F1
00F2:           00060 HIRESL   .EQ $F2
00F3:           00070 HIRESH   .EQ $F3
00F4:           00080 SL       .EQ $F4
00F5:           00090 SH       .EQ $F5
                00100 *OTHER EQUATES
6000:           00110 SCREEN   .EQ $6000 ;SCREEN IS ACTUALLY OFFSET 496 BYTES @ $61F0
6000:           00120 NDLIST   .EQ $6000 ;ADR OF DISPLAY LIST
0278:           00130 STICK    .EQ $278
02C5:           00140 COLOR1   .EQ $2C5
02C6:           00150 COLOR2   .EQ $2C6
D201:           00160 AUDC1    .EQ $D201
D200:           00170 AUDF1    .EQ $D200
00F6:           00180 NOTEL    .EQ $F6
00F7:           00190 NOTEH    .EQ $F7
E45C:           00200 SETVBK   .EQ $E45C
E462:           00210 XITVBK   .EQ $E462
```

```
                00220 *LINE LOOKUP TABLE - LO BYTE - STARTS AT $4000
4000: F0 18 40
4003: 68 90 B8
4006: E0 08     00230 YVERTL    .HS F018406890B8E008
4008: 30 58 80
400B: A8 D0 F8
400E: 20 48     00240           .HS 305880A8D0F82048
4010: 70 98 C0
4013: E8 10 38
4016: 60 88     00250           .HS 7098C0E810386088
4018: B0 D8 00
401B: 28 50 78
401E: A0 C8     00260           .HS B0D800285078A0C8
4020: F0 18 40
4023: 68 90 B8
4026: E0 08     00270           .HS F018406890B8E008
4028: 30 58 80
402B: A8 D0 F8
402E: 20 48     00280           .HS 305880A8D0F82048
4030: 70 98 C0
4033: E8 10 38
4036: 60 88     00290           .HS 7098C0E810386088
4038: B0 D8 00
403B: 28 50 78
403E: A0 C8     00300           .HS B0D800285078A0C8
```

BLIMP —FLOW CHART



```
4040: F0 18 40
4043: 68 90 B8
4046: E0 08     00310           .HS F018406890B8E008
4048: 30 58 80
404B: A8 D0 F8
404E: 20 48     00320           .HS 305880A8D0F82048
4050: 70 98 C0
4053: E8 10 38
4056: 60 88     00330           .HS 7098C0E810386088
4058: B0 D8 00
405B: 28 50 78
405E: A0 C8     00340           .HS B0D800285078A0C8
4060: F0 18 40
4063: 68 90 B8
4066: E0 08     00350           .HS F018406890B8E008
4068: 30 58 80
406B: A8 D0 F8
406E: 20 48     00360           .HS 305880A8D0F82048
4070: 70 98 C0
4073: E8 10 38
4076: 60 88     00370           .HS 7098C0E810386088
4078: B0 D8 00
407B: 28 50 78
407E: A0 C8     00380           .HS B0D800285078A0C8
4080: F0 18 40
4083: 68 90 B8
4086: E0 08     00390           .HS F018406890B8E008
4088: 30 58 80
408B: A8 D0 F8
408E: 20 48     00400           .HS 305880A8D0F82048
4090: 70 98 C0
4093: E8 10 38
4096: 60 88     00410           .HS 7098C0E810386088
4098: B0 D8 00
409B: 28 50 78
409E: A0 C8     00420           .HS B0D800285078A0C8
40A0: F0 18 40
40A3: 68 90 B8
40A6: E0 08     00430           .HS F018406890B8E008
40A8: 30 58 80
40AB: A8 D0 F8
40AE: 20 48     00440           .HS 305880A8D0F82048
40B0: 70 98 C0
40B3: E8 10 38
40B6: 60 88     00450           .HS 7098C0E810386088
40B8: B0 D8 00
40BB: 28 50 78
40BE: A0 C8     00460           .HS B0D800285078A0C8
40C0:           00470           .BS $40
                00480 *LINE LOOKUP TABLE - HI BYTE STARTS AT $4100
4100: 01 02 02
4103: 02 02 02
4106: 02 03     00490 YVERTH    .HS 0102020202020203
4108: 03 03 03
410B: 03 03 03
410E: 04 04     00500           .HS 0303030303030404
4110: 04 04 04
4113: 04 05 05
4116: 05 05     00510           .HS 0404040405050505
4118: 05 05 06
411B: 06 06 06
411E: 06 06     00520           .HS 0505060606060606
```

```
4120: 06 07 07
4123: 07 07 07
4126: 07 08     00530          .HS 0607070707070708
4128: 08 08 08
412B: 08 08 08
412E: 09 09     00540          .HS 0808080808080909
4130: 09 09 09
4133: 09 0A 0A
4136: 0A 0A     00550          .HS 090909090A0A0A0A
4138: 0A 0A 0B
413B: 0B 0B 0B
413E: 0B 0B     00560          .HS 0A0A0B0B0B0B0B0B
4140: 0B 0C 0C
4143: 0C 0C 0C
4146: 0C 0D     00570          .HS 0B0C0C0C0C0C0C0D
4148: 0D 0D 0D
414B: 0D 0D 0D
414E: 0E 0E     00580          .HS 0D0D0D0D0D0D0E0E
4150: 0E 0E 0E
4153: 0E 0F 0F
4156: 0F 0F     00590          .HS 0E0E0E0E0F0F0F0F
4158: 0F 0F 10
415B: 10 10 10
415E: 10 10     00600          .HS 0F0F101010101010
4160: 10 11 11
4163: 11 11 11
4166: 11 12     00610          .HS 1011111111111112
4168: 12 12 12
416B: 12 12 12
416E: 13 13     00620          .HS 1212121212121313
4170: 13 13 13
4173: 13 14 14
4176: 14 14     00630          .HS 1313131314141414
4178: 14 14 15
417B: 15 15 15
417E: 15 15     00640          .HS 1414151515151515
4180: 15 16 16
4183: 16 16 16
4186: 16 17     00650          .HS 1516161616161617
4188: 17 17 17
418B: 17 17 17
418E: 18 18     00660          .HS 1717171717171818
4190: 18 18 18
4193: 18 19 19
4196: 19 19     00670          .HS 1818181819191919
4198: 19 19 1A
419B: 1A 1A 1A
419E: 1A 1A     00680          .HS 19191A1A1A1A1A1A
41A0: 1A 1B 1B
41A3: 1B 1B 1B
41A6: 1B 1C     00690          .HS 1A1B1B1B1B1B1B1C
41A8: 1C 1C 1C
41AB: 1C 1C 1C
41AE: 1D 1D     00700          .HS 1C1C1C1C1C1C1D1D
41B0: 1D 1D 1D
41B3: 1D 1E 1E
41B6: 1E 1E     00710          .HS 1D1D1D1D1E1E1E1E
41B8: 1E 1E 1F
41BB: 1F 1F 1F
41BE: 1F 1F     00720          .HS 1E1E1F1F1F1F1F1F
41C0:           00730          .BS $40
```

```
                00740 *DISPLAY LIST STARTS AT $4200
4200: 70 70 70
4203: 4F F0 61
4206: 0F 0F     00750 DLIST     .HS 7070704FF0610F0F
4208: 0F 0F 0F
420B: 0F 0F 0F
420E: 0F 0F     00760           .HS 0F0F0F0F0F0F0F0F
4210: 0F 0F 0F
4213: 0F 0F 0F
4216: 0F 0F     00770           .HS 0F0F0F0F0F0F0F0F
4218: 0F 0F 0F
421B: 0F 0F 0F
421E: 0F 0F     00780           .HS 0F0F0F0F0F0F0F0F
4220: 0F 0F 0F
4223: 0F 0F 0F
4226: 0F 0F     00790           .HS 0F0F0F0F0F0F0F0F
4228: 0F 0F 0F
422B: 0F 0F 0F
422E: 0F 0F     00800           .HS 0F0F0F0F0F0F0F0F
4230: 0F 0F 0F
4233: 0F 0F 0F
4236: 0F 0F     00810           .HS 0F0F0F0F0F0F0F0F
4238: 0F 0F 0F
423B: 0F 0F 0F
423E: 0F 0F     00820           .HS 0F0F0F0F0F0F0F0F
4240: 0F 0F 0F
4243: 0F 0F 0F
4246: 0F 0F     00830           .HS 0F0F0F0F0F0F0F0F
4248: 0F 0F 0F
424B: 0F 0F 0F
424E: 0F 0F     00840           .HS 0F0F0F0F0F0F0F0F
4250: 0F 0F 0F
4253: 0F 0F 0F
4256: 0F 0F     00850           .HS 0F0F0F0F0F0F0F0F
4258: 0F 0F 0F
425B: 0F 0F 0F
425E: 0F 4F     00860           .HS 0F0F0F0F0F0F4F
4260: 00 70 0F
4263: 0F 0F 0F
4266: 0F 0F     00870           .HS 00700F0F0F0F0F0F
4268: 0F 0F 0F
426B: 0F 0F 0F
426E: 0F 0F     00880           .HS 0F0F0F0F0F0F0F0F
4270: 0F 0F 0F
4273: 0F 0F 0F
4276: 0F 0F     00890           .HS 0F0F0F0F0F0F0F0F
4278: 0F 0F 0F
427B: 0F 0F 0F
427E: 0F 0F     00900           .HS 0F0F0F0F0F0F0F0F
4280: 0F 0F 0F
4283: 0F 0F 0F
4286: 0F 0F     00910           .HS 0F0F0F0F0F0F0F0F
4288: 0F 0F 0F
428B: 0F 0F 0F
428E: 0F 0F     00920           .HS 0F0F0F0F0F0F0F0F
4290: 0F 0F 0F
4293: 0F 0F 0F
4296: 0F 0F     00930           .HS 0F0F0F0F0F0F0F0F
4298: 0F 0F 0F
429B: 0F 0F 0F
429E: 0F 0F     00940           .HS 0F0F0F0F0F0F0F0F
42A0: 0F 0F 0F
```

```
42A3: 0F 0F 0F
42A6: 0F 0F     00950           .HS 0F0F0F0F0F0F0F0F
42A8: 0F 0F 0F
42AB: 0F 0F 0F
42AE: 0F 0F     00960           .HS 0F0F0F0F0F0F0F0F
42B0: 0F 0F 0F
42B3: 0F 0F 0F
42B6: 0F 0F     00970           .HS 0F0F0F0F0F0F0F0F
42B8: 0F 0F 0F
42BB: 0F 0F 0F
42BE: 0F 0F     00980           .HS 0F0F0F0F0F0F0F0F
42C0: 0F 0F 0F
42C3: 0F 0F 0F
42C6: 0F 41     00990           .HS 0F0F0F0F0F0F0F41
42C8: 00 60     01000           .HS 0060
42CA:           01010           .BS $36
                01020 *HORIZONTAL OFFSET TABLE - START AT $4300
                01030 *THIS POINTS TO PROPER OFFSET SHAPE FOR EACH X POS (0-159)
4300: 00 01 02
4303: 03 00 01
4306: 02 03     01040 XOFF      .HS 0001020300010203
4308: 00 01 02
430B: 03 00 01
430E: 02 03     01050           .HS 0001020300010203
4310: 00 01 02
4313: 03 00 01
4316: 02 03     01060           .HS 0001020300010203
4318: 00 01 02
431B: 03 00 01
431E: 02 03     01070           .HS 0001020300010203
4320: 00 01 02
4323: 03 00 01
4326: 02 03     01080           .HS 0001020300010203
4328: 00 01 02
432B: 03 00 01
432E: 02 03     01090           .HS 0001020300010203
4330: 00 01 02
4333: 03 00 01
4336: 02 03     01100           .HS 0001020300010203
4338: 00 01 02
433B: 03 00 01
433E: 02 03     01110           .HS 0001020300010203
4340: 00 01 02
4343: 03 00 01
4346: 02 03     01120           .HS 0001020300010203
4348: 00 01 02
434B: 03 00 01
434E: 02 03     01130           .HS 0001020300010203
4350: 00 01 02
4353: 03 00 01
4356: 02 03     01140           .HS 0001020300010203
4358: 00 01 02
435B: 03 00 01
435E: 02 03     01150           .HS 0001020300010203
4360: 00 01 02
4363: 03 00 01
4366: 02 03     01160           .HS 0001020300010203
4368: 00 01 02
436B: 03 00 01
436E: 02 03     01170           .HS 0001020300010203
4370: 00 01 02
4373: 03 00 01
```

```
4376: 02 03     01180           .HS 0001020300010203
4378: 00 01 02
437B: 03 00 01
437E: 02 03     01190           .HS 0001020300010203
4380: 00 01 02
4383: 03 00 01
4386: 02 03     01200           .HS 0001020300010203
4388: 00 01 02
438B: 03 00 01
438E: 02 03     01210           .HS 0001020300010203
4390: 00 01 02
4393: 03 00 01
4396: 02 03     01220           .HS 0001020300010203
4398: 00 01 02
439B: 03 00 01
439E: 02 03     01230           .HS 0001020300010203
                01240 *SHPLO CONTAINS LO ORDER BYTE ADDRESS OF OUR 4 SHAPES
43A0: 00 17 2E
43A3: 45        01250 SHPLO     .HS 00172E45
                01260 *SHPHI CONTAINS HI ORDER BYTE ADDRESS OF OUR 4 SHAPES
43A4: 44 45 46
43A7: 47        01270 SHPHI     .HS 44454647
43A8:           01280           .BS $58
                01290 *SHAPES MUST BEGIN AT $4400
                01300 *SHAPE0
4400: 02 AA AA
4403: AA AA AA
4406: 80 00 00 01310 SHAPES     .HS 02AAAAAAAAAA800000
4409: 0A AA AA
440C: AA AA AA
440F: A8 00 00 01320            .HS 0AAAAAAAAAAAA80000
4412: 2A AA AA
4415: AA AA AA
4418: AA 00 00 01330            .HS 2AAAAAAAAAAAAA0000
441B: 2A AA AA
441E: AA AA AA
4421: AA 80 00 01340            .HS 2AAAAAAAAAAAAA8000
4424: AA AA AA
4427: AA AA AA
442A: AA 80 00 01350            .HS AAAAAAAAAAAAAA8000
442D: AA AA AA
4430: AA AA AA
4433: AA 80 00 01360            .HS AAAAAAAAAAAAAA8000
4436: 2A AA AA
4439: AA AA AA
443C: AA 80 00 01370            .HS 2AAAAAAAAAAAAA8000
443F: 2A AA AA
4442: AA AA AA
4445: AA 00 00 01380            .HS 2AAAAAAAAAAAAA0000
4448: 0A AA AA
444B: AA AA AA
444E: A8 00 00 01390            .HS 0AAAAAAAAAAAA80000
4451: 00 00 01
4454: 55 40 00
4457: 00 00 00 01400            .HS 000001554000000000
445A: 00 03 81
445D: 55 40 00
4460: 00 00 00 01410            .HS 000381554000000000
4463: 00 03 80
4466: 55 00 00
4469: 00 00 00 01420            .HS 000380550000000000
446C: 00 01 00
```

```
446F: 15 00 00
4472: 00 00 00 01430          .HS 000100150000000000
4475: 00 07 80
4478: 18 00 00
447B: 00 00 00 01440          .HS 000780180000000000
447E: 00 01 80
4481: 18 00 00
4484: 00 00 00 01450          .HS 000180180000000000
4487: 40 07 AA
448A: 8A 80 00
448D: C0 A0 00 01460          .HS 4007AA8A8000C0A000
4490: 50 1F 02
4493: AA 80 00
4496: C2 A8 00 01470          .HS 501F02AA8000C2A800
4499: 50 1C 00
449C: AA 80 00
449F: E2 AB 00 01480          .HS 501C00AA8000E2AB00
44A2: 54 00 00
44A5: 0A 80 00
44A8: EA AB 00 01490          .HS 5400000A8000EAAB00
44AB: 55 07 FF
44AE: FF FF FD
44B1: EA AB 00 01500          .HS 5507FFFFFFFDEAAB00
44B4: 55 0F FF
44B7: FF FF E5
44BA: E2 AB 00 01510          .HS 550FFFFFFFE5E2AB00
44BD: 55 5F FF
44C0: FF FF D4
44C3: 02 A8 00 01520          .HS 555FFFFFFFD402A800
44C6: 55 40 00
44C9: 18 00 10
44CC: 00 A0 00 01530          .HS 554000180010000A000
44CF: 50 50 00
44D2: 18 00 50
44D5: 00 00 00 01540          .HS 505000180050000000
44D8: 50 10 00
44DB: 18 00 40
44DE: 00 00 00 01550          .HS 501000180040000000
44E1: 40 14 00
44E4: 18 01 40
44E7: 00 00 00 01560          .HS 401400180140000000
44EA: 00 05 00
44ED: 18 01 00
44F0: 00 00 00 01570          .HS 000500180100000000
44F3: 00 05 40
44F6: 54 05 00
44F9: 00 00 00 01580          .HS 000540540500000000
44FC: 00 01 55
44FF: 55 54 00
4502: 00 00 00 01590          .HS 000155555400000000
4505: 00 00 55
4508: 55 54 00
450B: 00 00 00 01600          .HS 000055555400000000
450E: 00 00 15
4511: 01 50 00
4514: 00 00 00 01610          .HS 000015015000000000
                 01620 *SHAPE #1 SHIFTED RT 2 PIXELS
4517: 00 AA AA
451A: AA AA AA
451D: A0 00 00 01630          .HS 00AAAAAAAAAAA00000
4520: 02 AA AA
4523: AA AA AA
```

```
4526: AA 00 00 01640          .HS 02AAAAAAAAAAAA0000
4529: 0A AA AA
452C: AA AA AA
452F: AA 80 00 01650          .HS 0AAAAAAAAAAAAA8000
4532: 0A AA AA
4535: AA AA AA
4538: AA A0 00 01660          .HS 0AAAAAAAAAAAAAA000
453B: 2A AA AA
453E: AA AA AA
4541: AA A0 00 01670          .HS 2AAAAAAAAAAAAAA000
4544: 2A AA AA
4547: AA AA AA
454A: AA A0 00 01680          .HS 2AAAAAAAAAAAAAA000
454D: 0A AA AA
4550: AA AA AA
4553: AA A0 00 01690          .HS 0AAAAAAAAAAAAAA000
4556: 0A AA AA
4559: AA AA AA
455C: AA 80 00 01700          .HS 0AAAAAAAAAAAAA8000
455F: 02 AA AA
4562: AA AA AA
4565: AA 00 00 01710          .HS 02AAAAAAAAAAAA0000
4568: 00 00 00
456B: 55 50 00
456E: 00 00 00 01720          .HS 000000555000000000
4571: 00 00 E0
4574: 55 50 00
4577: 00 00 00 01730          .HS 0000E0555000000000
457A: 00 00 E0
457D: 15 40 00
4580: 00 00 00 01740          .HS 0000E0154000000000
4583: 00 00 40
4586: 05 40 00
4589: 00 00 00 01750          .HS 000040054000000000
458C: 00 01 E0
458F: 06 00 00
4592: 00 00 00 01760          .HS 0001E00600000000000
4595: 00 00 60
4598: 06 00 00
459B: 00 30 00 01770          .HS 000060060000003000
459E: 10 01 EA
45A1: A2 A0 00
45A4: 30 28 00 01780          .HS 1001EAA2A000302800
45A7: 14 07 C0
45AA: AA A0 00
45AD: 30 28 00 01790          .HS 1407C0AAA000302800
45B0: 14 07 00
45B3: AA A0 00
45B6: 30 AA C0 01800          .HS 140700AAA00030AAC0
45B9: 15 00 00
45BC: 02 A0 00
45BF: 78 AA C0 01810          .HS 15000002A00078AAC0
45C2: 15 01 FF
45C5: FF FF FF
45C8: 7A AA C0 01820          .HS 1501FFFFFFFF7AAAC0
45CB: 15 03 FF
45CE: FF FF F9
45D1: 78 AA C0 01830          .HS 1503FFFFFFF978AAC0
45D4: 15 57 FF
45D7: FF FF F5
45DA: 00 AA 00 01840          .HS 1557FFFFFFF500AA00
```

```
45DD: 15 50 00
45E0: 06 00 04
45E3: 00 28 00 01850          .HS 155000060004002800
45E6: 14 14 00
45E9: 06 00 14
45EC: 00 00 00 01860          .HS 141400060014000000
45EF: 14 04 00
45F2: 06 00 10
45F5: 00 00 00 01870          .HS 140400060010000000
45F8: 10 05 00
45FB: 06 00 50
45FE: 00 00 00 01880          .HS 100500060050000000
4601: 00 01 40
4604: 06 00 40
4607: 00 00 00 01890          .HS 000140060040000000
460A: 00 01 50
460D: 15 01 40
4610: 00 00 00 01900          .HS 000150150140000000
4613: 00 00 55
4616: 55 55 00
4619: 00 00 00 01910          .HS 000055555500000000
461C: 00 00 15
461F: 55 55 00
4622: 00 00 00 01920          .HS 000015555500000000
4625: 00 00 05
4628: 40 54 00
462B: 00 00 00 01930          .HS 000005405400000000
                 01940 *SHAPE #2 SHIFTER RT 4 PIXELS
462E: 00 2A AA
4631: AA AA AA
4634: A8 00 00 01950          .HS 002AAAAAAAAAA80000
4637: 00 AA AA
463A: AA AA AA
463D: AA 80 00 01960          .HS 00AAAAAAAAAAA8000
4640: 02 AA AA
4643: AA AA AA
4646: AA A0 00 01970          .HS 02AAAAAAAAAAAA000
4649: 02 AA AA
464C: AA AA AA
464F: AA A8 00 01980          .HS 02AAAAAAAAAAAA800
4652: 0A AA AA
4655: AA AA AA
4658: AA A8 00 01990          .HS 0AAAAAAAAAAAAA800
465B: 0A AA AA
465E: AA AA AA
4661: AA A8 00 02000          .HS 0AAAAAAAAAAAAA800
4664: 02 AA AA
4667: AA AA AA
466A: AA A8 00 02010          .HS 02AAAAAAAAAAAA800
466D: 02 AA AA
4670: AA AA AA
4673: AA A0 00 02020          .HS 02AAAAAAAAAAAA000
4676: 00 AA AA
4679: AA AA AA
467C: AA 80 00 02030          .HS 00AAAAAAAAAAA8000
467F: 00 00 00
4682: 15 54 00
4685: 00 00 00 02040          .HS 000000155400000000
4688: 00 00 38
468B: 15 54 00
468E: 00 00 00 02050          .HS 000038155400000000
```

```
4691: 00 00 38
4694: 05 50 00
4697: 00 00 00 02060          .HS 000038055000000000
469A: 00 00 10
469D: 01 50 00
46A0: 00 00 00 02070          .HS 000010015000000000
46A3: 00 00 78
46A6: 01 80 00
46A9: 00 00 00 02080          .HS 000078018000000000
46AC: 00 00 18
46AF: 01 80 00
46B2: 00 00 00 02090          .HS 000018018000000000
46B5: 04 00 7E
46B8: E8 A8 00
46BB: 0C 0A 00 02100          .HS 04007EE8A8000C0A00
46BE: 05 01 F0
46C1: 6A A8 00
46C4: 0C 2A 80 02110          .HS 0501F06AA8000C2A80
46C7: 05 01 C0
46CA: 0A A8 00
46CD: 1E 2A B0 02120          .HS 0501C00AA8001E2AB0
46D0: 05 40 00
46D3: 00 A8 00
46D6: 1E AA B0 02130          .HS 05400000A8001EAAB0
46D9: 05 50 7F
46DC: FF FF FF
46DF: DE AA B0 02140          .HS 05507FFFFFFFDEAAB0
46E2: 05 50 FF
46E5: FF FF FE
46E8: 5E 2A B0 02150          .HS 0550FFFFFFFE5E2AB0
46EB: 05 55 FF
46EE: FF FF FD
46F1: 40 2A 80 02160          .HS 0555FFFFFFFD402A80
46F4: 05 54 00
46F7: 01 80 01
46FA: 00 0A 00 02170          .HS 055400018001000A00
46FD: 05 05 00
4700: 01 80 05
4703: 00 00 00 02180          .HS 050500018005000000
4706: 05 01 00
4709: 01 80 04
470C: 00 00 00 02190          .HS 050100018004000000
470F: 04 01 40
4712: 01 80 14
4715: 00 00 00 02200          .HS 040140018014000000
4718: 00 00 50
471B: 01 80 10
471E: 00 00 00 02210          .HS 000050018010000000
4721: 00 00 54
4724: 05 40 50
4727: 00 00 00 02220          .HS 000054054050000000
472A: 00 00 15
472D: 55 55 40
4730: 00 00 00 02230          .HS 000015555540000000
4733: 00 00 05
4736: 55 55 40
4739: 00 00 00 02240          .HS 000005555540000000
473C: 00 00 01
473F: 50 15 00
4742: 00 00 00 02250          .HS 000001501500000000
              02260 *SHAPE #3 SHIFTED RT 6 PIXELS
```

```
4745: 00 0A AA
4748: AA AA AA
474B: AA 00 00 02270          .HS 000AAAAAAAAAAA0000
474E: 00 2A AA
4751: AA AA AA
4754: AA A0 00 02280          .HS 002AAAAAAAAAAA000
4757: 00 AA AA
475A: AA AA AA
475D: AA A8 00 02290          .HS 00AAAAAAAAAAAA800
4760: 00 AA AA
4763: AA AA AA
4766: AA AA 00 02300          .HS 00AAAAAAAAAAAAA00
4769: 02 AA AA
476C: AA AA AA
476F: AA AA 00 02310          .HS 02AAAAAAAAAAAAA00
4772: 02 AA AA
4775: AA AA AA
4778: AA AA 00 02320          .HS 02AAAAAAAAAAAAA00
477B: 00 AA AA
477E: AA AA AA
4781: AA AA 00 02330          .HS 00AAAAAAAAAAAAA00
4784: 00 AA AA
4787: AA AA AA
478A: AA A8 00 02340          .HS 00AAAAAAAAAAAA800
478D: 00 2A AA
4790: AA AA AA
4793: AA A0 00 02350          .HS 002AAAAAAAAAAA000
4796: 00 00 00
4799: 05 55 00
479C: 00 00 00 02360          .HS 000000055500000000
479F: 00 00 0E
47A2: 05 55 00
47A5: 00 00 00 02370          .HS 00000E055500000000
47A8: 00 00 0E
47AB: 01 54 00
47AE: 00 00 00 02380          .HS 00000E015400000000
47B1: 00 00 04
47B4: 00 54 00
47B7: 00 00 00 02390          .HS 000004005400000000
47BA: 00 00 1E
47BD: 00 60 00
47C0: 00 00 00 02400          .HS 00001E006000000000
47C3: 00 00 06
47C6: 00 60 00
47C9: 00 00 00 02410          .HS 000006006000000000
47CC: 01 00 1E
47CF: AA 2A 00
47D2: 03 02 80 02420          .HS 01001EAA2A00030280
47D5: 01 40 7C
47D8: 0A AA 00
47DB: 03 0A A0 02430          .HS 01407C0AAA00030AA0
47DE: 01 40 70
47E1: 02 AA 00
47E4: 07 8A AC 02440          .HS 01407002AA00078AAC
47E7: 01 50 00
47EA: 00 2A 00
47ED: 07 AA AC 02450          .HS 015000002A0007AAAC
47F0: 01 54 1F
47F3: FF FF FF
47F6: F7 AA AC 02460          .HS 01541FFFFFFFF7AAAC
47F9: 01 54 3F
```

```
47FC: FF FF FF
47FF: 97 8A AC 02470          .HS 01543FFFFFFF978AAC
4802: 01 55 7F
4805: FF FF FF
4808: 50 0A A0 02480          .HS 01557FFFFFFF500AA0
480B: 01 55 00
480E: 00 60 00
4811: 40 02 80 02490          .HS 015500006000400280
4814: 01 41 40
4817: 00 60 01
481A: 40 00 00 02500          .HS 014140006001400000
481D: 01 40 40
4820: 00 60 01
4823: 00 00 00 02510          .HS 014040006001000000
4826: 01 00 50
4829: 00 60 05
482C: 00 00 00 02520          .HS 010050006005000000
482F: 00 00 14
4832: 00 60 04
4835: 00 00 00 02530          .HS 000014006004000000
4838: 00 00 15
483B: 01 50 14
483E: 00 00 00 02540          .HS 000015015014000000
4841: 00 00 05
4844: 55 55 50
4847: 00 00 00 02550          .HS 000005555550000000
484A: 00 00 01
484D: 55 55 50
4850: 00 00 00 02560          .HS 000001555550000000
4853: 00 00 00
4856: 54 05 40
4859: 00 00 00 02570          .HS 000000540540000000
              02580 *START TABLE ON EVEN PAGE BOUNDARY
485C:         02590           .BS $A4
4900: 43 10 3F
4903: 08 38 20
4906: 00 14    02600 NOTES    .HS 43103F0838200014 ;P1M1
4908: 43 08 4B
490B: 10 43 08
490E: 54 40    02610          .HS 43084B1043085440
4910: 43 10 43
4913: 08 4B 10
4916: 54 08    02620          .HS 431043084B105408
4918: 65 20 00
491B: 10 65 08
491E: 43 10    02630          .HS 6520001065084310 ;P1M2
4920: 43 08 4B
4923: 20 00 10 02640          .HS 43084B200010
4926: 43 10 3F
4929: 08 38 20
492C: 00 14    02650          .HS 43103F0838200014 ;P1M1
492E: 43 08 4B
4931: 10 43 08
4934: 54 40    02660          .HS 43084B1043085440
4936: 43 10 43
4939: 08 4B 10
493C: 54 08    02670          .HS 431043084B105408
493E: 65 20 00
4941: 10 65 08
4944: 43 10    02680          .HS 6520001065084310 ;P1M2
4946: 43 08 4B
```

```
4949: 20 00 10 02690          .HS 43084B200010
494C: 38 20 00
494F: 02 38 20
4952: 00 02 38
4955: 20 38 10 02700          .HS 382000023820000238203810 ;P2M1
4958: 00 02 38
495B: 10 32 08
495E: 4B 10 00
4961: 02       02710          .HS 0002381032084B100002
4962: 4B 08 00
4965: 02 4B 10
4968: 00 02 4B
496B: 08 00 02
496E: 4B 40 00
4971: 02 4B 10 02720          .HS 4B0800024B1000024B0800024B4000024B10 ;P2M2
4974: 00 02 4B
4977: 08 00 02
497A: 4B 10 00
497D: 02 4B 08
4980: 00 02 4B
4983: 40       02730          .HS 00024B0800024B1000024B0800024B40
4984: 54 10 00
4987: 02 54 08
498A: 00 02 54
498D: 10 00 02
4990: 54 08 00
4993: 02 54 40 02740          .HS 541000025408000254100002540800025440 ;P2M3
4996: 38 20 00
4999: 02 38 20
499C: 00 02 38
499F: 20 38 10 02750          .HS 382000023820000238203810 ;P2M1
49A2: 00 02 38
49A5: 10 32 08
49A8: 4B 10 00
49AB: 02       02760          .HS 0002381032084B100002
49AC: 4B 08 00
49AF: 02 4B 10
49B2: 00 02 4B
49B5: 08 00 02
49B8: 4B 40 00
49BB: 02 4B 10 02770          .HS 4B0800024B1000024B0800024B4000024B10 ;P2M2
49BE: 00 02 4B
49C1: 08 00 02
49C4: 4B 10 00
49C7: 02 4B 08
49CA: 00 02 4B
49CD: 40       02780          .HS 00024B0800024B1000024B0800024B40
49CE: 54 10 00
49D1: 02 54 08
49D4: 00 02 54
49D7: 10 00 02
49DA: 54 08 00
49DD: 02 54 40 02790          .HS 541000025408000254100002540800025440 ;P2M3
49E0: 00 15 FF 02800          .HS 0015FF ;REST & REPEAT
               02810 *VARIABLES
49E3:          02820 X        .BS 1
49E4:          02830 Y        .BS 1
49E5:          02840 HORIZ    .BS 1
49E6:          02850 TEMP     .BS 1
49E7:          02860 DEPTH    .BS 1
49E8:          02870 SLNGH    .BS 1
```

```
49E9:           02880 VERT     .BS 1
49EA:           02890 OFFSET   .BS 1
49EB:           02900 POINTER  .BS 1
49EC:           02910 TIME     .BS 1
49ED:           02920 TDELAY   .BS 1
49EE:           02925 VBFLAG   .BS 1
                02930 *CLEAR SCREEN 8K INCLUDING NDLIST AREA
49EF: A9 00     02940 CLRSCR   LDA #SCREEN ;SETUP POINTERS TO CLEAR SCREEN
49F1: 85 F4     02950          STA SL
49F3: A9 60     02960          LDA /SCREEN
49F5: 85 F5     02970          STA SH
49F7: A0 00     02980          LDY #$00
49F9: 98        02990          TYA
49FA: A2 20     03000          LDX #$20    ;32 PAGES (8K)
49FC: 91 F4     03010 .2       STA (SL),Y
49FE: C8        03020          INY
49FF: D0 FB     03030          BNE .2      ;CONTINUE UNTIL DONE WITH 256 BYTES
4A01: E6 F5     03040          INC SH      ;DO NEXT PAGE
4A03: CA        03050          DEX
4A04: D0 F6     03060          BNE .2
                03070 *SETUP DLIST
4A06: A2 00     03080          LDX #$00
4A08: BD 00 42  03090 DLOOP    LDA DLIST,X
4A0B: 9D 00 60  03100          STA NDLIST,X
4A0E: E8        03110          INX
4A0F: E0 CC     03120          CPX #$CC    ;204 ELEMENTS
4A11: D0 F5     03130          BNE DLOOP
4A13: A9 00     03140          LDA #NDLIST
4A15: 8D 30 02  03150          STA 560
4A18: A9 60     03160          LDA /NDLIST
4A1A: 8D 31 02  03170          STA 561
                03180 *INITILIZE
4A1D: A9 0B     03190          LDA #$0B
4A1F: 8D C5 02  03200          STA COLOR1
4A22: A9 00     03210          LDA #$00
4A24: 8D C6 02  03220          STA COLOR2
4A27: A9 40     03230          LDA #$40
4A29: 8D E3 49  03240          STA X
4A2C: A9 20     03250          LDA #$20
4A2E: 8D E4 49  03260          STA Y
4A31: 8D E9 49  03270          STA VERT
4A34: A9 00     03280          LDA #$00
4A36: 8D ED 49  03290          STA TDELAY
4A39: A9 00     03300          LDA #NOTES  ;GET LO BYTE OF TABLE
4A3B: 85 F6     03310          STA NOTEL
4A3D: A9 49     03320          LDA /NOTES  ;GET HI BYTE OF TABLE
4A3F: 85 F7     03330          STA NOTEH
4A41: A9 00     03340          LDA #$00    ;CLEAR TIMER
4A43: 85 14     03350          STA $14
4A45: A9 01     03360          LDA #$01
4A47: 8D EC 49  03370          STA TIME
4A4A: A9 EA     03380          LDA #$EA    ;PURE TONE - VOLUME 10
4A4C: 8D 01 D2  03390          STA AUDC1
4A4F: A0 00     03400          LDY #$00
4A51: 8C EB 49  03410          STY POINTER
                03420 *SET VBLANK
4A54: A9 07     03430          LDA #$07
4A56: A2 4A     03440          LDX /FRAME
4A58: A0 60     03450          LDY #FRAME
4A5A: 20 5C E4  03460          JSR SETVBK
4A5D: 4C 5D 4A  03465 FOREVER  JMP FOREVER
```

```
                    03470 *VBLANK ROUTINE
4A60: AD EE 49      03472 FRAME    LDA VBFLAG
4A63: F0 03         03473          BEQ OK
4A65: 4C 62 E4      03474          JMP $E462
4A68: A9 01         03475 OK       LDA #$01
4A6A: 8D EE 49      03476          STA VBFLAG
4A6D: AD E3 49      03480          LDA X          ;CALC NEW HORIZ OFFSET
4A70: 4A            03490          LSR            ;DIVIDE BY 4 TO GET HORIZ BYTE
4A71: 4A            03500          LSR
4A72: 8D E5 49      03510          STA HORIZ
4A75: 20 E7 4A      03520          JSR SETUP
4A78: 20 0B 4B      03530          JSR DRAW       ;DRAW SHAPE
4A7B: AD ED 49      03540          LDA TDELAY     ;DELAY 3/60 TH SEC
4A7E: C9 03         03550          CMP #$03
4A80: D0 1F         03560          BNE .1
4A82: 20 E7 4A      03570          JSR SETUP
4A85: 20 90 4B      03580          JSR XDRAW      ;XDRAW SHAPE
4A88: 20 45 4B      03590          JSR JOYSTK     ;READ JOYSTICK
4A8B: AD E3 49      03600          LDA X          ;CALC NEW HORIZ OFFSET
4A8E: 4A            03610          LSR
4A8F: 4A            03620          LSR
4A90: 8D E5 49      03630          STA HORIZ
4A93: 20 E7 4A      03640          JSR SETUP
4A96: 20 0B 4B      03650          JSR DRAW       ;IMMEDIATELY REDRAW SHAPE
4A99: A9 00         03660          LDA #$00       ;RESET DELAY
4A9B: 8D ED 49      03670          STA TDELAY
4A9E: 4C A4 4A      03680          JMP SOUND
4AA1: EE ED 49      03690 .1       INC TDELAY
4AA4: AD EC 49      03700 SOUND    LDA TIME
4AA7: C5 14         03710          CMP $14
4AA9: B0 34         03720          BGE .2
                    03730 *STOP NOTE - EQUIVALENT TO SOUND 0,0,0,0
4AAB: A9 00         03740          LDA #$00
4AAD: 8D 08 D2      03750          STA $D208
4AB0: A9 03         03760          LDA #$03
4AB2: 8D 0F D2      03770          STA $D20F
4AB5: AC EB 49      03780          LDY POINTER
                    03790 *LOAD NEXT NOTE
4AB8: B1 F6         03800          LDA (NOTEL),Y
4ABA: C9 FF         03810          CMP #$FF       ;LAST NOTE?
4ABC: D0 09         03820          BNE .3
                    03830 *RESET TO BEGINNING NOTE
4ABE: A0 00         03840          LDY #$00
4AC0: 8C EB 49      03850          STY POINTER
4AC3: A9 49         03860          LDA /NOTES
4AC5: 85 F7         03870          STA NOTEH
                    03880 *CONTINUE READING NOTES & STORING VALUES IN SOUND REGISTERS
4AC7: 8D 00 D2      03890 .3       STA AUDF1      ;STORE FREQUENCY OF NEXT NOTE
4ACA: C8            03900          INY
4ACB: EE EB 49      03910          INC POINTER
4ACE: B1 F6         03920          LDA (NOTEL),Y
4AD0: 8D EC 49      03930          STA TIME       ;STORE LENGTH OF NOTE
4AD3: C8            03940          INY
4AD4: EE EB 49      03950          INC POINTER
4AD7: D0 02         03960          BNE .1         ;CAN ONLY HAPPEN ON AN EVEN NUMBER
4AD9: E6 F7         03970          INC NOTEH      ;NEXT PAGE OF NOTES
4ADB: A9 00         03980 .1       LDA #$00       ;CLEAR TIMER AT BEGIINING OF EACH NEW NOTE
4ADD: 85 14         03990          STA $14
4ADF: A9 00         03995 .2       LDA #$00
4AE1: 8D EE 49      03996          STA VBFLAG
4AE4: 4C 62 E4      04000          JMP XITVBK
```

```
                04010 *SUBROUTINES
                04020 *DRAWING SETUP SUBROUTINE
4AE7: AC E3 49  04030 SETUP     LDY X        ;HORIZONTAL POSITION (0-159)
4AEA: BE 00 43  04040           LDX XOFF,Y   ;INDEX TO FIND SHAPE #
4AED: BD A0 43  04050           LDA SHPLO,X ;INDEX TO GET LO BYTE OF SHAPE TABLE
4AF0: 85 F0     04060           STA SHPL     ;STORE LO BYTE IN ZERO PAGE
4AF2: BD A4 43  04070           LDA SHPHI,X ;GET HI BYTE OF SHAPE TABLE
4AF5: 85 F1     04080           STA SHPH
4AF7: AD E4 49  04090           LDA Y
4AFA: 8D E9 49  04100           STA VERT
4AFD: A9 1F     04110           LDA #$1F
4AFF: 8D E7 49  04120           STA DEPTH    ;SHAPE IS 31 LINES DEEP
4B02: A9 09     04130           LDA #$09
4B04: 8D E8 49  04140           STA SLNGH    ;SHAPE IS 9 BYTES WIDE
4B07: 8D E6 49  04150           STA TEMP     ;STORED HERE ALSO BECAUSE DRAWING
                                              ROUTINE DECREMENTS SLNGH 0
                04160 *  ;AND VARIABLE MUST BE RESTORED AT START OF NEXT ROW
4B0A: 60        04170           RTS
                04180 *DRAW SHAPE SUBROUTINE
4B0B: AC E9 49  04190 DRAW      LDY VERT     ;VERTICAL POSITION
4B0E: 20 34 4B  04200           JSR GETADR   ;FIND BEGINNING OF SCREEN ADDRESS OF ROW
4B11: A2 00     04210           LDX #$00
4B13: AD E6 49  04220           LDA TEMP
4B16: 8D E8 49  04230           STA SLNGH    ;RESTORE VALUE OF WIDTH FOR NEXT ROW
4B19: A0 00     04240           LDY #$00
4B1B: A1 F0     04250 DRAW2     LDA (SHPL,X);GET BYTE OF SHAPE TABLE
4B1D: 91 F2     04260           STA (HIRESL),Y ;PLOT ON SCREEN
4B1F: E6 F0     04270           INC SHPL     ;NEXT BYTE OF SHAPE TABLE
4B21: D0 02     04280           BNE .1       ;IF CROSS PAGE BOUNDARY?
4B23: E6 F1     04290           INC SHPH     ;INCREMENT TO NEXT PAGE OF SHAPE
4B25: C8        04300 .1        INY          ;NEXT POSITION ON SCREEN
4B26: CE E8 49  04310           DEC SLNGH    ;DECREMENT WIDTH
4B29: D0 F0     04320           BNE DRAW2    ;FINISHED WITH ROW YET
4B2B: EE E9 49  04330           INC VERT     ;IF SO, INCREMENT TO NEXT LINE
4B2E: CE E7 49  04340           DEC DEPTH    ;DECREMENT DEPTH
4B31: D0 D8     04350           BNE DRAW     ;FINISHED ALL ROWS?
4B33: 60        04360           RTS          ;YES, END
                04370 *GETADR SUBROUTINE FOR FINDING STARTING SCREEN ADDRESS
                       TO PLOT BYTES
4B34: B9 00 40  04380 GETADR    LDA YVERTL,Y ;LOOKUP LO BYTE OF LINE
4B37: 18        04390           CLC
4B38: 6D E5 49  04400           ADC HORIZ    ;ADD HORIZ OFFSET
4B3B: 85 F2     04410           STA HIRESL   ;STORE LO BYTE SCREEN ADDRESS
4B3D: B9 00 41  04420           LDA YVERTH,Y ;LOOKUP HI BYTE LINE
4B40: 69 60     04430           ADC /SCREEN ;ADD HI BYTE OF SCREEN
4B42: 85 F3     04440           STA HIRESH   ;STORE HI BYTE SCREEN ADDRESS
4B44: 60        04450           RTS
                04460 *JOYSTICK ROUTINE
4B45: AD 78 02  04470 JOYSTK    LDA STICK
4B48: 29 02     04480           AND #$02     ;DOWN BIT?
4B4A: D0 0D     04490           BNE CHKLF
4B4C: AD E4 49  04500           LDA Y        ;PREVENT SHAPE FROM EXITING BOTTOM SCREEN
4B4F: C9 A0     04510           CMP #$A0
4B51: F0 06     04520           BEQ CHKLF
4B53: EE E4 49  04530           INC Y        ;MOVE TWO LINES
4B56: EE E4 49  04540           INC Y
4B59: AD 78 02  04550 CHKLF     LDA STICK
4B5C: 29 04     04560           AND #$04     ;LEFT BIT?
4B5E: D0 0A     04570           BNE CHKRT
4B60: AD E3 49  04580           LDA X        ;PREVENT SHAPE FROM EXITING SCREEN LEFT
4B63: C9 00     04590           CMP #$00
4B65: F0 03     04600           BEQ CHKRT
```

```
4B67: CE E3 49 04610          DEC X       ;THIS MOVES TWO PIXELS
4B6A: AD 78 02 04620 CHKRT    LDA STICK
4B6D: 29 08    04630          AND #$08    ;RIGHT BIT?
4B6F: D0 0A    04640          BNE CHKUP
4B71: AD E3 49 04650          LDA X       ;PREVENT SHAPE FROM EXITING SCREEN RIGHT
4B74: C9 7C    04660          CMP #$7C
4B76: F0 03    04670          BEQ CHKUP
4B78: EE E3 49 04680          INC X
4B7B: AD 78 02 04690 CHKUP    LDA STICK
4B7E: 29 01    04700          AND #$01    ;UP BIT
4B80: D0 0D    04710          BNE .1
4B82: AD E4 49 04720          LDA Y       ;PREVENT SHAPE FROM EXITING TOP SCREEN
4B85: C9 00    04730          CMP #$00
4B87: F0 06    04740          BEQ .1
4B89: CE E4 49 04750          DEC Y
4B8C: CE E4 49 04760          DEC Y
4B8F: 60       04770 .1       RTS
               04780 *XDRAW SHAPE SUBROUTINE
4B90: AC E9 49 04790 XDRAW    LDY VERT    ;VERTICAL POSITION
4B93: 20 34 4B 04800          JSR GETADR  ;FIND BEGINNING OF SCREEN ADDRESS OF ROW
4B96: A2 00    04810          LDX #$00
4B98: AD E6 49 04820          LDA TEMP
4B9B: 8D E8 49 04830          STA SLNGH   ;RESTORE VALUE OF WIDTH FOR NEXT ROW
4B9E: A0 00    04840          LDY #$00
4BA0: A1 F0    04850 XDRAW2   LDA (SHPL,X);GET BYTE OF SHAPE TABLE
4BA2: 51 F2    04860          EOR (HIRESL),Y ;EOR WITH SCREEN IMAGE
4BA4: 91 F2    04870          STA (HIRESL),Y ;PLOT ON SCREEN
4BA6: E6 F0    04880          INC SHPL    ;NEXT BYTE OF SHAPE TABLE
4BA8: D0 02    04890          BNE .1      ;IF CROSS PAGE BOUNDARY?
4BAA: E6 F1    04900          INC SHPH    ;INCREMENT TO NEXT PAGE OF SHAPE
4BAC: C8       04910 .1       INY         ;NEXT POSITION ON SCREEN
4BAD: CE E8 49 04920          DEC SLNGH   ;DECREMENT WIDTH
4BB0: D0 EE    04930          BNE XDRAW2  ;FINISHED WITH ROW YET
4BB2: EE E9 49 04940          INC VERT    ;IF SO, INCREMENT TO NEXT LINE
4BB5: CE E7 49 04950          DEC DEPTH   ;DECREMENT DEPTH
4BB8: D0 D6    04960          BNE XDRAW
4BBA: 60       04970          RTS         ;YES, END
```

Sound

Sound complements graphics in nearly all arcade-style games. While most people think of sound effects as the only necessary sound, the addition of an original background score can contribute greatly to a game's overall popularity. In either case, the Atari, with its four-voice sound chip, is well-suited to the task.

The Atari computer has four independent voices that can vary in pitch by more than three octaves. The tone can vary from very pure to highly distorted. In addition, each voice has its own loudness level, completely independent of the television's volume setting.

BASIC's Sound Statment

In BASIC, the SOUND statement takes the following form:

SOUND Voice, Pitch, Distortion, Loudness

The first parameter Voice is simple. There are four voices or channels whose numbers range from 0-3. It takes a separate sound statement to activate each channel. Initially, at least in BASIC, they are all off at any time, but any one can be selectively turned off by setting Pitch, Distortion, and Loudness for that voice to all zeros.

Pitch can vary between 0 -255. The value 'N' is used in a divide circuit. For every N pulses coming in, one pulse goes out. As N gets larger, the output pulses become less frequent and make a lower note. A value of 121 produces a middle C tone. A pitch of 60 produces a C tone one octave higher, and a pitch of 243 produces a C tone one octave lower. Pitch values around 3 approach the edge of human hearing and may not be audible on a television speaker that lacks a tweeter.

The Atari computer produces both pure and distorted tones. The term distortion is actually a misnomer. All of the sound waves on the Atari are square waves. Distortion doesn't occur because of a degradation of the wave form like in harmonic audio, but by selectively removing pulses from the waveform. A more appropriate term would be noise. Distortion values of 10 and 14 generate pure tones. Other even-numbered distortion values (0,2,4,6, and 12) introduce different amounts of noise into the pure tone. The quality of the sound depends on both the pitch and the distortion. Some combinations, mainly distortion 12, combine to produce an undistorted secondary tone with harmonic overtones.

Loudness is controlled by the fourth number in the SOUND statement. The value varies from 0 (silent) to 15 (loudest) and is fairly linear for a single voice. The apparent loudness is affected by pitch. High-pitched sounds seem quieter than low-pitched sounds. If you are working with multi-channels, the sum of all four channels should not exceed thirty-two or it will overmodulate the audio output. The sound produced tends to actually lose volume and assume a buzzing quality.

Sound Duration

Since the SOUND statement lacks a duration parameter, sound can be turned on and then off by using an empty FOR...NEXT loop as a delay. It is largely experimental but empty FOR...NEXT loops iterate at approximately 450 times per second. A loop that goes from 1-225 would cause a delay of half a second. Thus, the following three lines would turn on a tone, let it sound for one-half second, then turn it off.

```
100 SOUND 0,121,10,10
110 FOR I=1 TO 225:NEXT I
120 SOUND 0,0,0,0
```

Sound Effects

Simple sound effects are created largely by trial and error. Many use FOR...NEXT loops to either vary the pitch or vary the volume. Some do both. The pistol sound in the blocks game in Chapter 5 varies the volume. The bonk sound of the brick being

removed is similar but at another low pitch. Both sounds use distortion or noise to achieve their effect.

```
100 REM - PISTOL SOUND
110 FOR L=10 TO 4 STEP -0.25
120 SOUND 0,10,0,L
130 NEXT L
```

```
100 REM - BONK SOUND FOR KNOCKING OUT BRICK
110 FOR L=15 TO 0 STEP -0.5
120 SOUND 0,20,2,L
130 NEXT L
```

It is also possible to vary both the pitch and the volume simultaneously in a loop. The following example simulates the sound of a falling bomb. It begins with a high pitch and gradually changes to a low pitch, followed by the thumping sound of an explosion.

```
100 REM - FALLING OBJECT
110 FOR L=30 TO 200 STEP 3
120 SOUND 0,L,10,L/25
130 FOR K=1 TO L/10:NEXT K
140 NEXT L
150 SOUND 0,20,0,14
160 SOUND 1,255,10,15
170 FOR K=1 TO 150:NEXT K
180 SOUND 1,0,0,0
```

Most sound effects have to be placed in a larger loop with the graphics or player-missile commands, or motion will stop while the sound routine runs. The problem is that this method often alters the time delays and preset durations of the sound effects. Worse yet, the location of these routines within the program changes the result. This occurs because BASIC must search its line number list whenever it encounters a branch or GOTO instruction. Obviously, it finds line numbers at the beginning of the program before it finds line numbers near the end. The only real solution to the problem is to run your sound routines in the VBlank period, and this approach requires Machine language programming skills.

Sound—Assembly Language

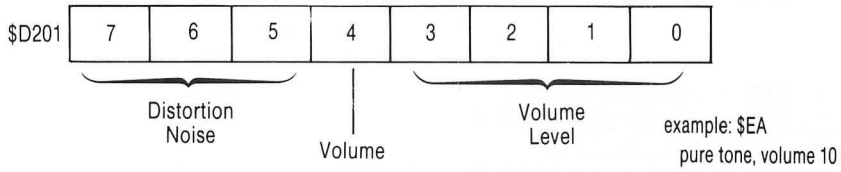
The POKEY digital I/O chip controls the audio frequency and the audio control registers for all four sound channels. The AUDF# (audio frequency) locations are used to control pitch, and the AUDC# (audio control) locations are used to control distortion and volume. The sound locations are as follows, and they are write registers only:

AUDF1 = $D000	AUDC1 = $D001
AUDF2 = $D002	AUDC2 = $D003
AUDF3 = $D004	AUDC3 = $D005
AUDF4 = $D006	AUDC4 = $D007

Frequency values range from $00 to $FF. POKEY actually increments this number by one before sending it to its divide by "N" circuit. For every N pulses coming in, one pulse comes out. Thus, the higher the value of N, the lower the tone. The rate of the pulses depends on the POKEY clock.

AUDC1—4 Sound Registers

The AUDC1-4 locations control both distortion and volume. The bit pattern is as follows:



The lower four bits control the volume level (0-15). Zero means no volume, while 15 means as loud as possible. The only constraint here is that the total volume for all four sound channels does not exceed thirty-two. Bit 4 is a volume-only control. Turning this bit on will force the speaker cone out. Trouble is that this by itself won't produce a tone since a tone is produced by repeatedly forcing the cone in and out rapidly. This bit can be useful to advanced sound programmers.

The upper three bits control the distortion. Distortion is produced by first dividing the clock value by the frequency, then masking the output using the various poly counters specified by the bit pattern. The result is finally divided by two. Poly counters or polynomial counters are actually shift registers that produce various degrees of distortion in random but repeatable sequences. Since they are repeatable, they are predictable and are useful for generating sound effects. In general, the tones become more regular or recognizable with fewer and lower poly counters masking the output. The 17-bit poly counter is useful for white noise effects like a waterfall, while the 4-bit poly counter is useful for a motor sound.

BIT 7	6	5	
0	0	0	5-bit, then 17-bit polys
0	0	1	5-bit poly only
0	1	0	5-bit, then 4-bit polys
0	1	1	5-bit poly only
1	0	0	17-bit poly only
1	0	1	no poly counters (pure tone)
1	1	0	4-bit poly only
1	1	1	oo poly counters (pure tone)

AUDCTL Register

In addition to the independent channel control bytes (AUDC1-4), there is one other register, AUDCTL at 53768 or $D208, that affects all of the channels. Each bit in AUDCTL is assigned a specific function.

Bit	Description
7	Makes the 17-bit poly counter into a 9-bit poly
6	Clock channel one with 1.79 MHz
5	Clock channel three with 1.79 MHz
4	Join channels one and two (16 bit)
3	Join channels three and four (16 bits)
2	Insert high pass filter into channel one, clocked by channel two
1	Insert high pass filter into channel two, clocked by channel four
0	Switch main clock base from 64 KHz to 15 KHz

Shifting the 17-bit poly counters to 9-bit poly counters by setting bit 7, will create more repeatable sound patterns rather than white noise-type patterns. Setting the channels to a higher clock frequency (setting bits 5 and 6), will produce higher tones. Likewise, setting the bit 7 from 64 KHz to 15 KHz will produce much lower tones.

If you couple two of the sound channels by setting either bits 3 or 4, you reduce the number of channels to two but gain increased tonal range. Normally, you get a five octave range using the eight bits of a single channel, but the combined 16-bit register increases the tonal range to nine octaves.

Sometimes you may encounter problems POKEing sounds in BASIC or in Machine language without initializing the sound registers. BASIC requires a null sound statement, i.e., SOUND 0,0,0,0. In Machine language you need to store a 0 at AUDCTL ($D208), and a 3 at SKCTL ($D20F).

Background Music

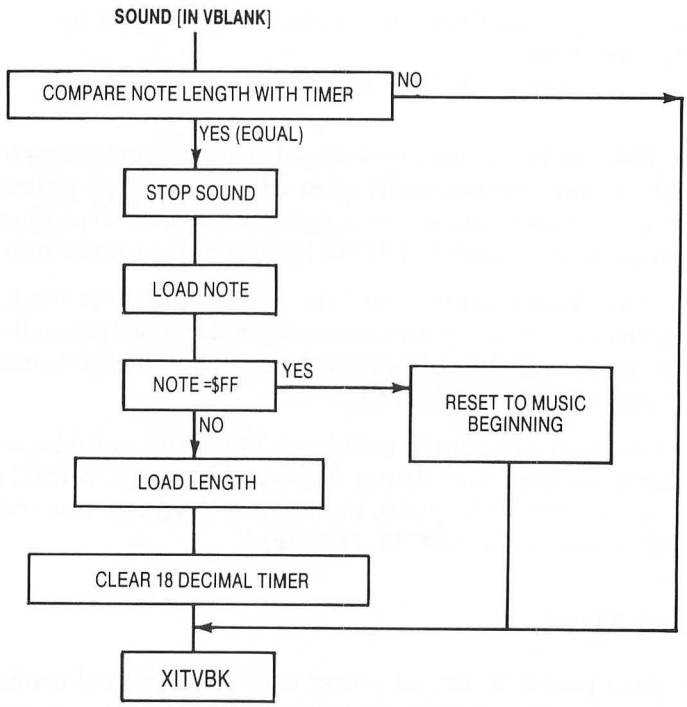
One of the most pleasing uses of sound is to play musical tunes quietly in the background, during many games or at least use them to enhance an animated title page. Such routines normally run in the Vertical Blank period so that the note lengths remain accurate. Generally, you store the notes and durations of the tune in a table. The Atari reads the note and its corresponding duration from the table. It then turns on the note and sustains it until the timer at $14, which counts in jiffies, reaches the value set by the duration. At that point the note shuts off and the computer reads the next note and duration. Two consecutive notes with the same pitch sound like they run together as a much longer note. It is often necessary to place a zero pitch lasting two jiffies between the notes. With this method, it is very simple to play an entire musical score without affecting the play mechanics or speed of a game. Be careful that you don't use and reset that timer elsewhere in the game.

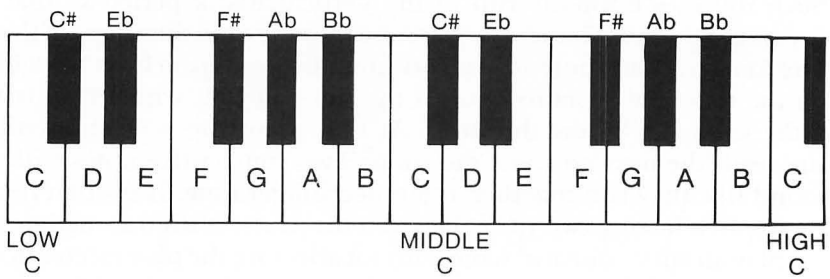
We use the value of $FF for the note as a flag to indicate the tune is finished. While it could be used to conclude the piece, we use it to reset the pointers to the table so that the music repeats endlessly.

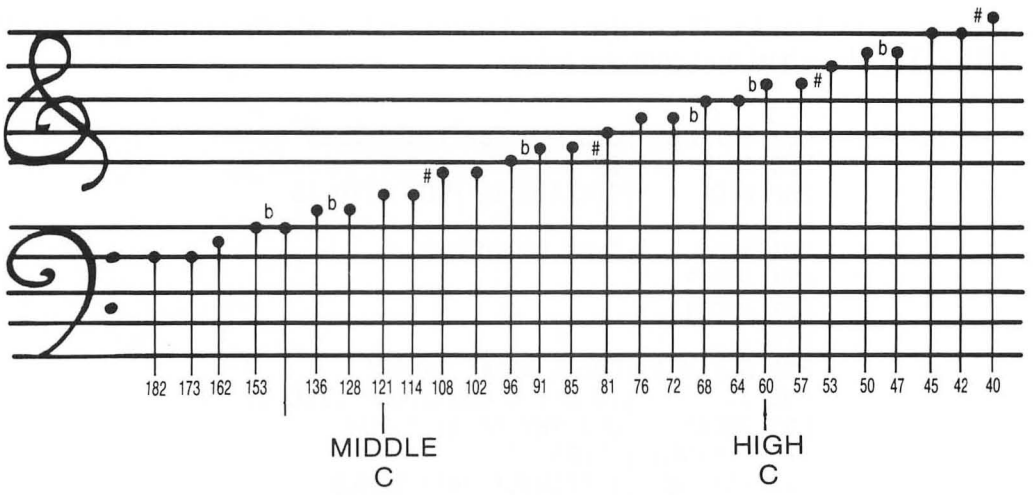
The lengths of the different notes is summarized in the table below:

NOTE	JIFFIES
Sixteenth	8
Eighth	15
Quarter	30
Half	60
Rest	60

code here (see code on blimp example page?)







Sound Effects

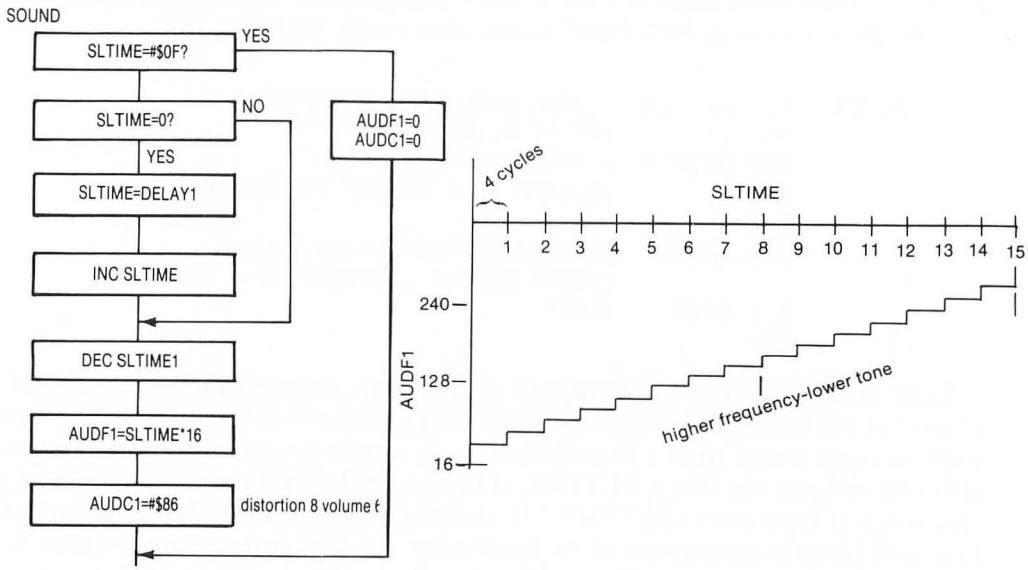
Explosion sounds are simple to implement in Machine language within the Vertical Blank routine. Basically, you need a very irregular rumbling sound that slowly decreases in volume. Setting the distortion to zero sets up 17-bit poly counters that produce quite irregular sound. The duration of the sound is controlled by a timer that counts down every jiffy. This timer can also control the volume level so that it decreases as a function of the value of the timer. For instance, if the sound is to take one second, SEXTIME, short for Set Explosion Timer, is initially set to 64 and is decremented every jiffy. If VOLUME = SEXTIME / 4, then the volume will decrease from 16 to 0 as SEXTIME counts down to 0. The code follows:

```
SOUND3   LDA SEXTIME ;CHECK EXPLOSION TIMER FLAG
         BEQ .1      ;IF AT 0, NO SOUND
         DEC SEXTIME ;COUNTDOWN
         LSR         ;DIVIDE BY 4 TO GET VOLUME 16-0
         LSR
         STA AUDC4   ;TELL POKEY NEW SOUND VOLUME
*                    ;UPPER NIBBLE (DISTORTION = 0)
         LDA #$40    ;TONE
.1       RTS
```

Laser fire can be simulated by rapidly changing the frequency from a high pitch to a lower one in discontinuous jumps while using a distortion set at 6. This produces a more staccato sound than a smooth frequency transition. You can implement this effect by making the timer, SLTIME, short for Set Laser Timer, a function of the frequency. If Frequency = SLTIME * 16, then each time SLTIME is incremented, the tone will jump in increments of 16. Remember, the higher the N in the divide by N circuit, the lower the tone. The problem here is that the sound is much too short if allowed to increment simply from 1 to 15. Therefore, a secondary loop delays each

tonal jump by 4 cycles. The entire sound routine takes 60 jiffies rather than just 15 jiffies. The flowchart and code are below:

```
SOUND   LDA SLTIME   ;CHECK LASER TIMER FLAG
        BEQ .3       ;IF 0 EXIT
        CMP #$OF     ;TIMER GOES FROM 1 TO 15
        BNE .1
        LDA #$OO     ;TURN SOUND OFF
        STA AUDF1
        STA AUDC1
        RTS
.1      LDA SLTIME1 ;CHECK DELAY TIMER
        BNE .2       ;IF NOT 0 COUNTDOWN TILL IT IS
        LDA DELAY1   ;GET NEW DELAY VALUE
        STA SLTIME1 ;STORE IT
        INC SLTIME   ;INCREMENT MAIN TIMER
;                    ;(THIS IS ALSO OUR FREQUENCY VALUE)
.2      DEC SLTIME1 ;OUR FREQUENCY VALUE
        ASL          ;MULTIPLY BY 16
        ASL
        ASL
        ASL
        STA AUDF1    ;NEW TONE VALUE
        LDA #$86     ;DISTORTION 8, VOLUME 6
        STA AUDC1
.3      RTS
```



CHAPTER 9

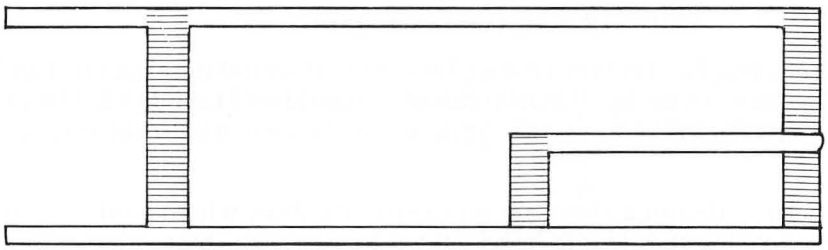
ADVANCED ARCADE TECHNIQUES

Maze Games

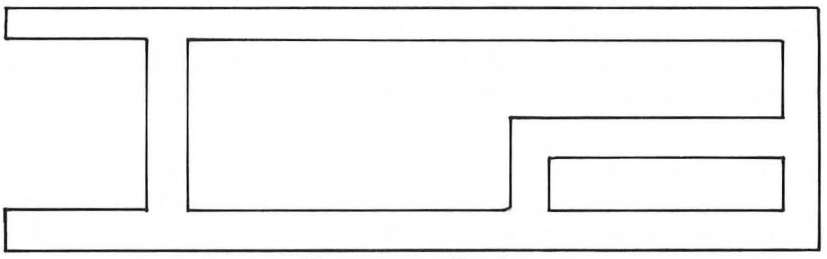
Maze games achieved the height of popularity with the debut of eat-the-dot games like *Pacman* and *Ms Pacman*. They weren't the first eat-the dots games: that honor goes to a car game called *Head-on*. They were, however, the first games to transpose the usual open field chase or pursuit games to the narrow confine of maze corridors.

Topographically, a maze is merely a network of interconnecting paths. These pathways constrain the player's movement. In a sense each individual section of the maze gives the player a set of movement rules. The passage walls that are open allow the player to reach the next section, while the closed walls block movement in other directions.

The maze can be divided into a number of small blocks, each the height and width of the passageway. Depending on the graphics mode, each of these blocks could assume the size of a character. That way each character becomes one of the blocks in the maze. The characters can be open blocks, with various combinations of open exterior walls, or they can be floors and ladders for use in a climbing, jumping arcade game. While most people do not think of games like *Donkey Kong* and *Apple Panic* as maze games, they too require a set of movement rules to keep the player confined to floors and ladders.



Floor & Ladder Blocks



Passageway Blocks

The instructions that guide a player about the maze can be very complicated, or they can be quite simple. They can take thousands of bytes or just several hundred. Naturally, each individual block needs to be a part of the playfield and requires one byte per block. If we drew our maze in graphics one characters (eight dots by eight rows), we could have a maze twenty blocks wide by twelve rows deep. That takes 240 bytes of memory. Next we need instructions to tell the player if he can move up, down, left or right from the center of the block where he is. That could take as many as four bytes per block. These could all be placed in tables so that if we knew which block we were in, we could index into each table look up the legal moves for that block.

Storing a zero for an open pathway and a one for a closed pathway doesn't use a memory location to its capacity. Only the lowest bit is used. If we could combine all four directions into one byte, with each direction using one bit, we would only use a fourth as much memory and still have half of the byte left. With Left using the lowest bit or 0th bit, Right the first bit, Up the second, and Down the third, we can test each of these bits by ANDing with a MASK that contains a 1 bit in the bit position we wish to test and 0 bits everywhere else.

LEFT	CLOSED	0000 0001
RIGHT	CLOSED	0000 0010
UP	CLOSED	0000 0100
DOWN	CLOSED	0000 1000

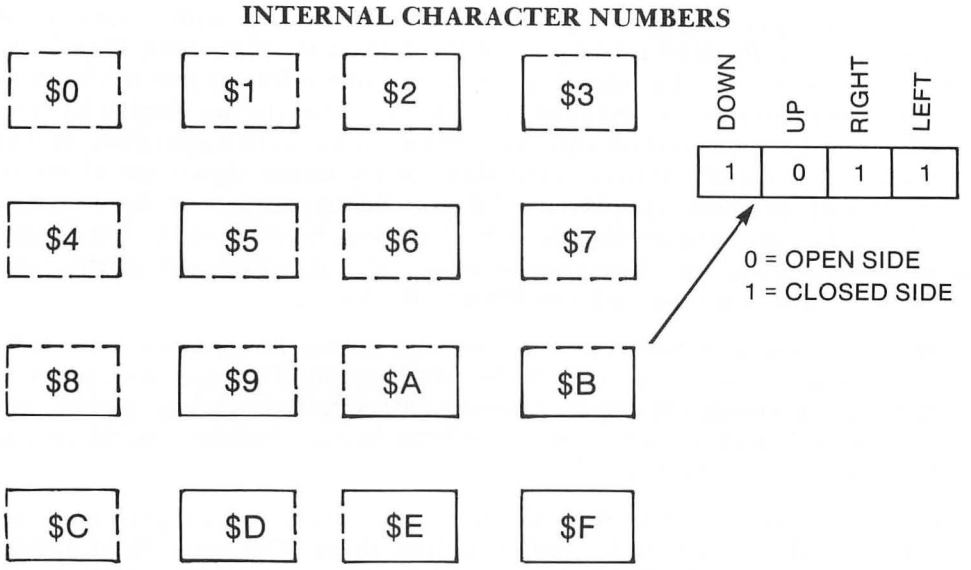
Thus to test if the up direction is closed we AND the instruction byte with $04. In the following example only the right direction is open.

```
0 0 0 0 1 1 0 1   INSTRUCTION BYTE
0 0 0 0 0 1 0 0   AND #$04
-----------------
0 0 0 0 0 1 0 0   RESULT IS POSITIVE IF
                  UP DIRECTION IS CLOSED
```

We could set a flag for the gates that we find open. If we did the above test and found the result positive, or the up direction closed, we could set FLAGU = 1. The four flags FLAGU, FLAGD, FLAGL, and FLAGR, would be set to 0 if found open and to 1 if closed.

Since we need to design a character set to visually show which walls are open and which ones are closed, it would be beneficial to mirror the byte value of the block's instruction byte. Thus, if a character had the only the left wall closed, its instruction byte would have a value of one. Therefore, we will design the first character in the character set to have only the left wall closed. The 0th character has no walls closed, and the fifteenth character in our table has all four walls closed. The instruction byte that reflects this has a value of 15 ($0F). Each of the characters in our set is shown below.

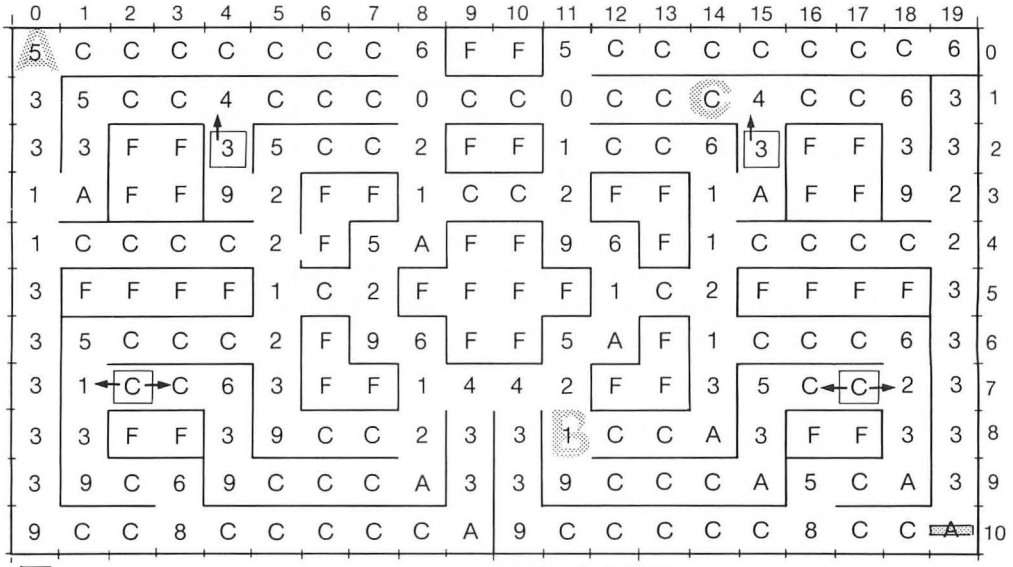
The playfield consists of twelve rows, each with twenty of these characters. As a whole they make up an entire maze. A player in the top left hand block in the maze is in the 0th position of the character data that ANTIC uses to generate the playfield or maze. If the player moves one row down to position XB,YB (0,1) it is at the twentieth



position of the character data; and one position to the right (1,1), the twenty-first position position in the data. This can be formalized as:

BLOCK = (YB*20) + XB

If we index into the screen data, SCREEN,Y where the Y register contains the value of the character at the player's position. That number is the same as the sum of the individual instruction bits for legal movement in each of the four directions. The fact that only one table is needed for both the screen data and the legal movement instructions is not a coincidence. It is simply a clever way to condense data.



9 ADVANCED ARCADE TECHNIQUES

The design of any maze game should include a simple premise with a simple set of rules. The object of *PacMan* is to eat all of the dots on the maze floor in order to advance to the next level. The object of the maze game is for the player's letter to chase two higher letters in the alphabet. The player catches the next higher letter in order to become that letter as it advances toward the letter Z and a harder maze. Just as the four ghosts are the adversaries in *PacMan*, the red minus sign is the adversary here. It constantly pursues your player and if it catches it, your letter value decreases by one. Thus, the game begins with the letter A chasing letters B and C. If it catches the B, it becomes a B and continues chasing a C and D. If the minus sign catches your player at that point, it reverts back one letter to the letter A.

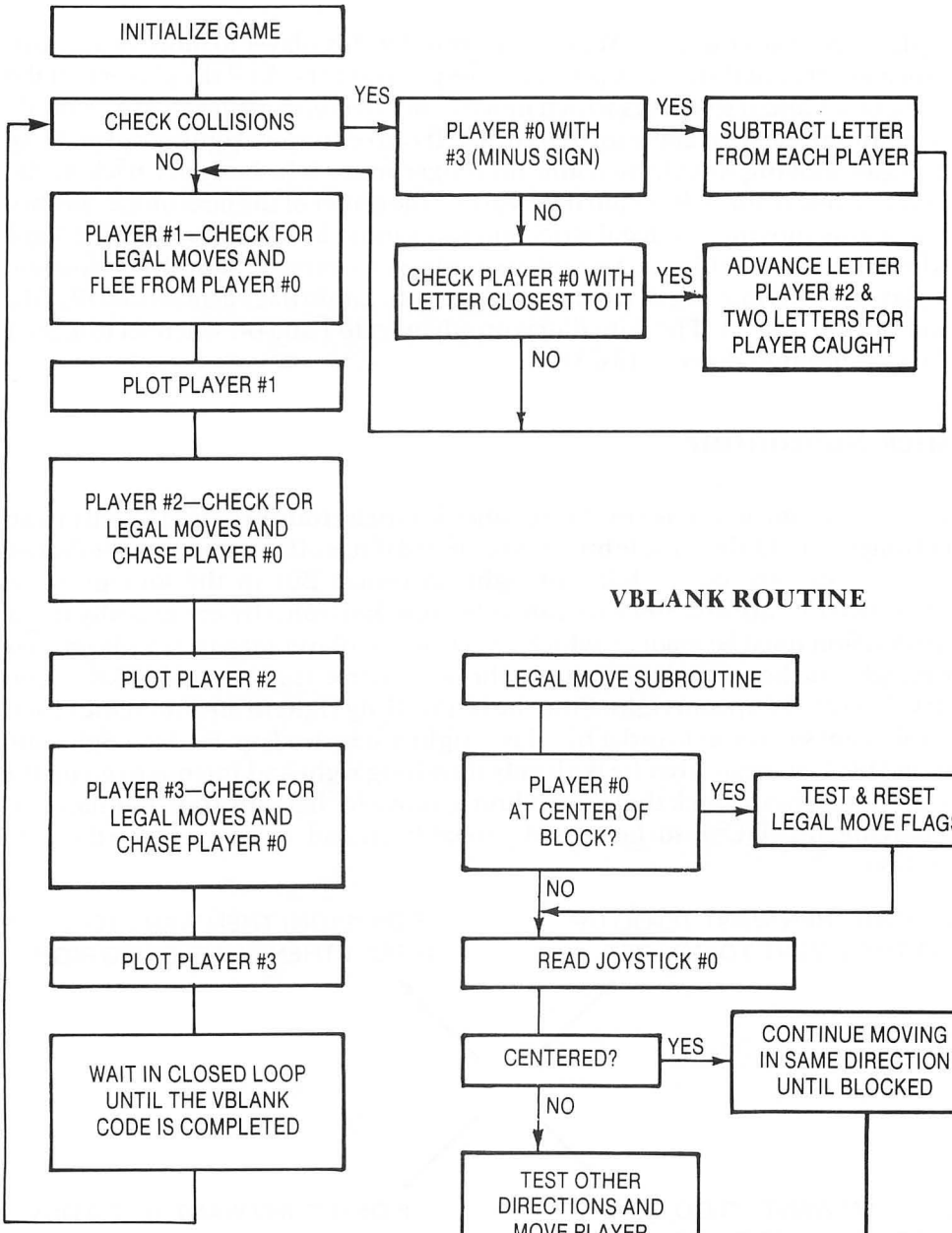
There is no point scorekeeping system for this game. The progress you make during the game becomes a visual scorekeeping system. The goal is to progress forward, not to gain points. If we were to award 100 points for each letter gained, and 50 points for each letter lost, players might prolong the game indefinitely, advancing several letters than losing a few.

All the players move at exactly the same speed. This forces strategic play, since you can't catch the fleeing letters by simply tailing them. The game must appear winnable, so we never make the player start the game over when he is caught by the minus sign. The game would become frustrating if a player advanced to the letter T and reverted back to an A just because the minus sign caught him repeatedly. Beginners would also find the game frustrating if the minus sign caught them and put them in another random position in the maze, just as they were about to trap a letter.

The maze game code is much too long to run within the time frame provided by vertical blank. Since I had tested my theory of maze game logic with a single player within the vertical blank period, I decided to place the remainder of the game code outside the vertical blank time, but sychronized with it. Normally code entirely within VBLANK runs in a endless loop (FOREVER JMP FOREVER) and only leaves this loop during the interrupt period. When your code runs outside VBLANK it runs until the VBLANK interupts it, then executes the code in the interupt routine. At the end of the vertical blank period it returns to where it left off in the code. With the 6502 runing at 1.8 MHz a moderately short program is likely to cycle through twenty or thirty passes between VBLANK interupts. The only solution to synchronize the two pieces of code is to allow the code to run once then wait in a tight loop testing for some flag that can only be set if the program reaches the vertical blank routine. This is precisely what we did. Our VBLANK routine sets a flag call VBFLAG = 1. The main program code sits in an endless loop testing this flag. It can only exit the loop to the beginning of the program code when VBFLAG = 1.

```
FOREVER  LDA VBFLAG   ;TEST FLAG
         CMP #$01
         BNE .1
         JMP LOOPM    ;EXITS WHEN VBFLAG=1
.1       JMP FOREVER
```

MAZE GAME OVERALL FLOWCHART



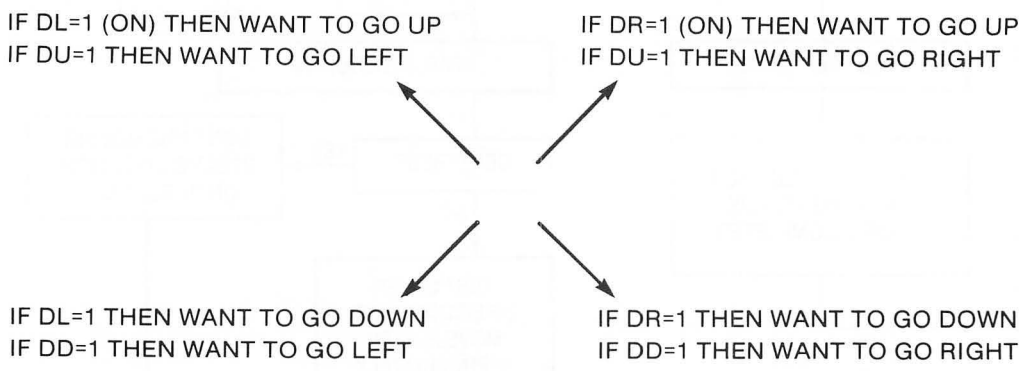
VBLANK ROUTINE

Player Movement

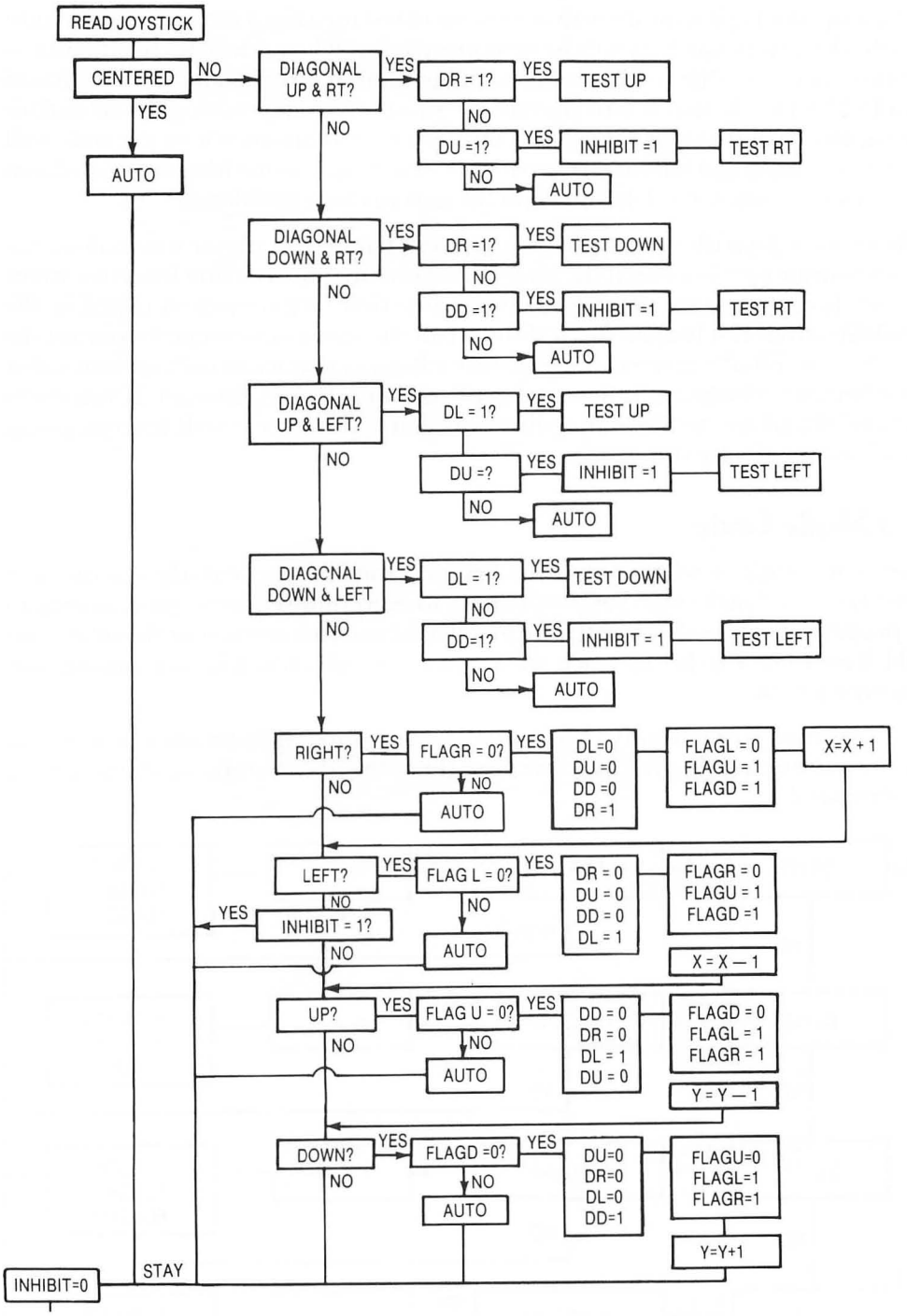
The player is joystick guided. When a player orders his player to move in a certain direction a number of things happen. First the program checks if the player is at the center of a block and if so checks which movement directions are legal. For example if a player initially started at the top left corner, it can only move right or down. If the player began moving down, he could no longer move left, but only back in the direction he came from, at least until he reached the center of the next block. So once a player begins moving the legal direction flags must be reset to reflect the legal moves between adjacent blocks. Second, since player movement should be automatic once a player begins moving in the desired direction, a auto flags denoted as DR, DL, DU, and DD must be set. The auto flag is on when set to 1 and off when set to 0. So if our player is heading downwards, DD=1.

Joystick Subroutine

The joystick subroutine is similar to other joysticks routines except in its treatment of diagonals. Only a single bit has to be tested if it is off to determine the desired direction for the up, down, left and right positions. But in the four diagonal directions two bits are off. Since we can only move horizontally or vertically in the maze, a decision must be made to which direction the player means. If a player were moving right and he wanted to turn up at the next intersection he would likely point his stick diagonally up and right. Since he is travelling right in an automatic mode, he probably doesn't mean to order his player right but instead up. So if the right auto flag is on DR=1, it means that he is already travelling right and intends to go up the next time he reaches a block that allows him to move in that direction. Similarily, if he is moving up, DU=1 so he would probably intend to go right at the next intersection.



If you look at the flow chart for the joystick subroutine, you will notice that in addition to setting auto flags and reseting legal flags for each of the four primary joystick directions, a flag called INHIBIT is set sometimes in the diagonal test. This is necessary because if for example we had an up and right diagonal and we were


READ JOYSTICK
CENTERED
NO
YES
AUTO
DIAGONAL UP & RT?
YES
NO
DR = 1?
YES
TEST UP
NO
DU =1?
YES
INHIBIT =1
TEST RT
NO
AUTO
DIAGONAL DOWN & RT?
YES
NO
DR =1?
YES
TEST DOWN
NO
DD =1?
YES
INHIBIT =1
TEST RT
NO
AUTO
DIAGONAL UP & LEFT?
YES
NO
DL = 1?
YES
TEST UP
DU =?
YES
INHIBIT =1
TEST LEFT
NO
AUTO
DIAGONAL DOWN & LEFT
YES
NO
DL = 1?
YES
TEST DOWN
NO
DD=1?
YES
INHIBIT = 1
TEST LEFT
NO
AUTO
RIGHT?
YES
NO
FLAGR = 0?
YES
NO
AUTO
DL=0
DU =0
DD =0
DR =1
FLAGL = 0
FLAGU = 1
FLAGD = 1
X=X + 1
LEFT?
YES
NO
FLAG L = 0?
YES
NO
AUTO
DR = 0
DU = 0
DD = 0
DL = 1
FLAGR = 0
FLAGU = 1
FLAGD =1
X = X — 1
INHIBIT = 1?
YES
NO
UP?
YES
NO
FLAG U = 0?
YES
NO
AUTO
DD = 0
DR = 0
DL = 1
DU = 0
FLAGD = 0
FLAGL = 1
FLAGR = 1
Y = Y — 1
DOWN?
YES
NO
FLAGD =0?
YES
NO
AUTO
DU=0
DR=0
DL=0
DD=1
FLAGU=0
FLAGL=1
FLAGR=1
Y=Y+1
STAY
INHIBIT=0

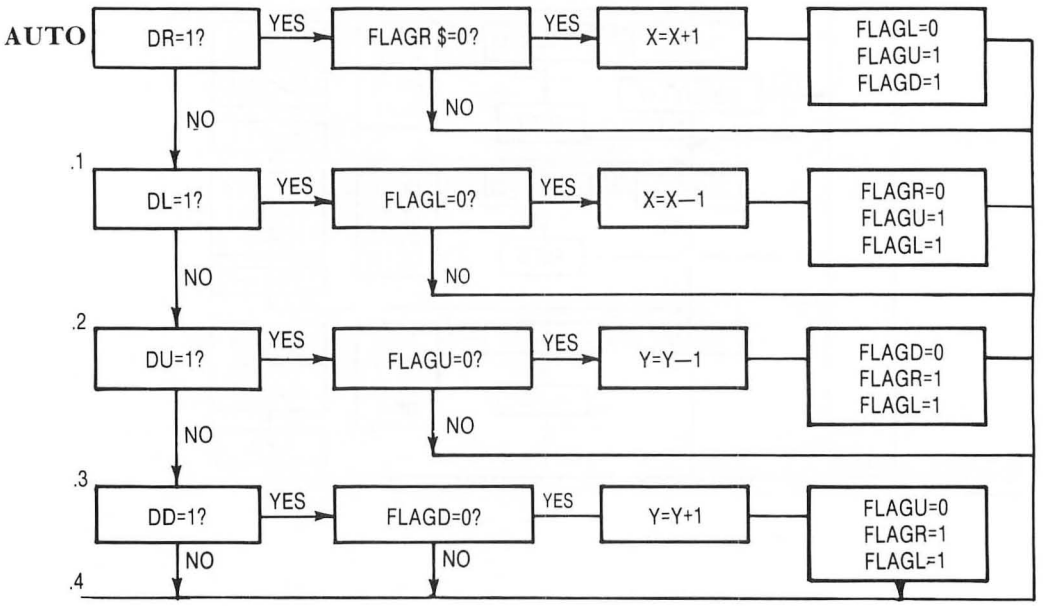
traveling up, the logic would say that we want to test for a right move. Since the right bit is off, the apropriate fags will be set correctly, but when it reaches the up bit test that bit is also off so that it will try to set latches for that direction too. But if it has to pass a INHIBIT = 1? test, it will prevent the program from reaching the second bit position test. The INHIBIT flags are only set for situations where the code will branch to the right and left bit tests first. Code that branches to either the up or down bit tests can not reach the left and right bit tests to cause problems.

Whenever a joystick command is given to change the player's direction, the joystick routine checks to see if the player can move in that direction from its current position. It checks to see if the legal move direction flag is open or closed in the desired direction. If it is open (0), it shuts off all the auto flags except the one for the new direction. Finally it resets the legal move flags so that it can only go forward or reverse between blocks, and then moves the player one unit forward. If you try to command the player to move in a direction that it can't go, it will keep on going forward automatically if it can.

Auto Mode Code

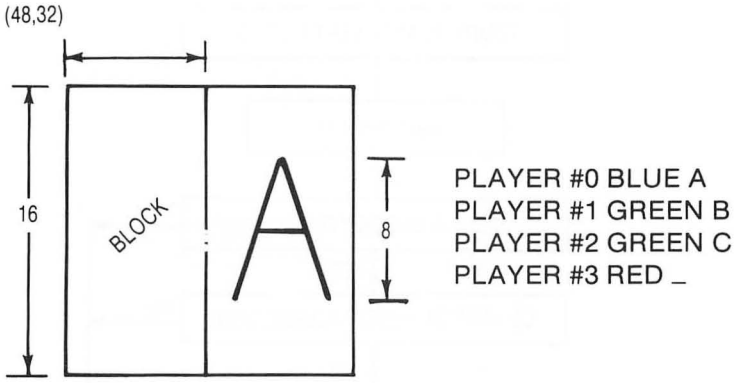
The auto mode is what makes the joystick routine particularly smooth and responsive. If it didn't exist, you would have to exert more effort to get your man to steer properly. Besides always needing to push the joystick one way or the other, you would have to be careful to reach the exact center of a block before successfully negoiating a turn.

The auto mode code simply checks to see which auto flag is on and checks if it is legal to continue moving in that direction. If not the player stops until it receives a new command.



Legal Move Subroutine

A subroutine appropriately called LEGAL determines in which directions the player is allowed to move. You input the player # in the X-register and its current X and Y positions, and it sets the legal move flags FLAGL,X , FLAGR,X , FLAGU,X , and FLAGD,X for the approprate player. It is a two step process. First it decides if the player is at the center of a block. Each block is 8 pixels wide by 16 pixels deep. For a player to be at the center of a block it must be at an exact multiple of 8 horizontally and an exact multiple of 16 vertically. The player's position is in player-missile coordinates. The top left corner of the maze is at location 48,32. And the 8 scan line high player is initially 4 scan lines lower in order to center it in a 16 scan line high



block. We can get two values TEMPX and TEMPY by subtracting 48 and 36 from the player's horizontal and vertical positions respectively. The coordinates of the block XB,YB that the player is currently in are calculated as follows.

```
TEMPX = XP-48
    XB = TEMPX/8

TEMPY = YP-36
    YB = TEMPY/16

BLOCK = YB*20 + XB
```

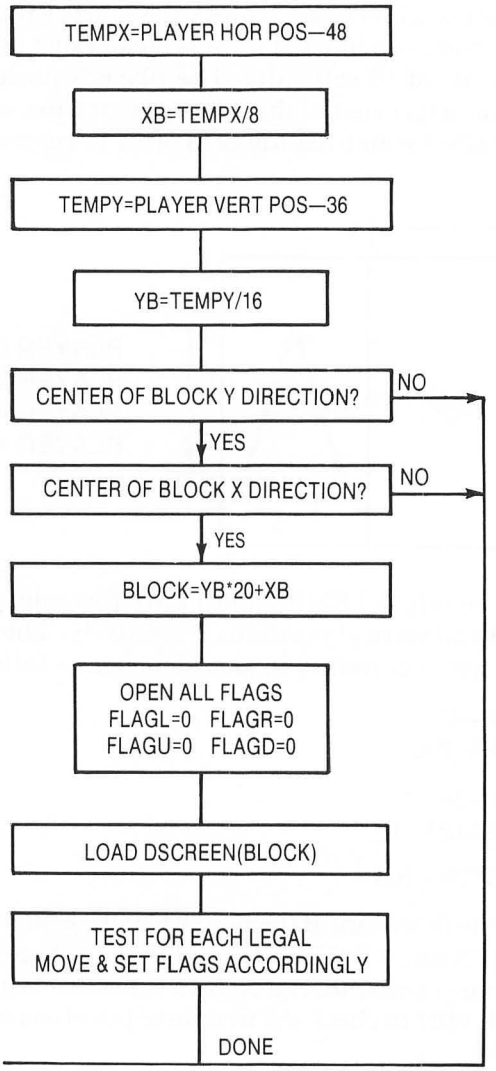
It is quite simple to determine if the player is at the center of the block by checking the values in TEMPX and TEMPY. If any of the first three bit positions in TEMPX containing anything (a remainder) we have a value that is not an exact multiple of 8. You can AND with #$07 to check the first three positions of TEMPX for a non zero value. For example:

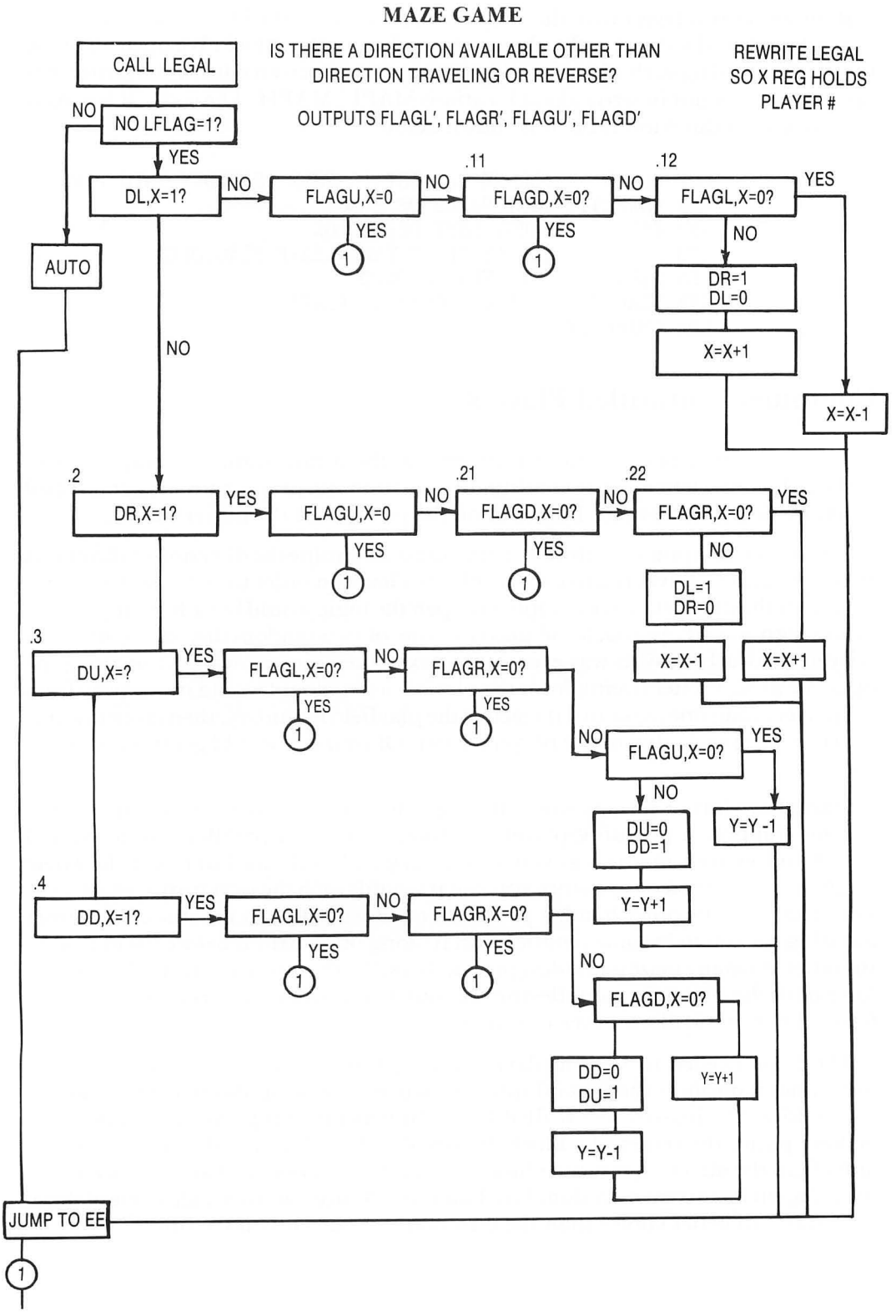
```
0 0 0 0 1 0 0 1   TEMPX = #$09
0 0 0 0 0 1 1 1   AND #$07
-----------------
0 0 0 0 0 0 0 1   RESULT positive
```

Similarly we can AND with #$0F to test the first four positions of TEMPY for a non-zero value or a remainder.

(SUBROUTINE LEGAL)

INPUT XP,YP PLAYER POSITIONS
OUTPUT FLAGL, FLAGR, FLAGU, FLAGD
0—OPEN 1—CLOSE




MAZE GAME
CALL LEGAL
IS THERE A DIRECTION AVAILABLE OTHER THAN DIRECTION TRAVELING OR REVERSE?
OUTPUTS FLAGL', FLAGR', FLAGU', FLAGD'
REWRITE LEGAL SO X REG HOLDS PLAYER #
NO
NO LFLAG=1?
YES
DL,X=1?
NO
FLAGU,X=0
NO
.11
FLAGD,X=0?
NO
.12
FLAGL,X=0?
YES
YES
YES
YES
1
1
AUTO
NO
DR=1
DL=0
X=X+1
X=X-1
.2
DR,X=1?
YES
FLAGU,X=0
NO
.21
FLAGD,X=0?
NO
.22
FLAGR,X=0?
YES
YES
YES
YES
1
1
NO
DL=1
DR=0
.3
DU,X=?
YES
FLAGL,X=0?
NO
FLAGR,X=0?
X=X-1
X=X+1
YES
YES
1
1
NO
FLAGU,X=0?
YES
NO
DU=0
DD=1
Y=Y -1
.4
DD,X=1?
YES
FLAGL,X=0?
NO
FLAGR,X=0?
Y=Y+1
YES
YES
1
1
NO
FLAGD,X=0?
DD=0
DU=1
Y=Y+1
Y=Y-1
JUMP TO EE
1

If the above two tests prove that we are at the center of the blocks, we can test the individual direction bits for the character number for that block. We open all of the flags before testing each direction. The address to the screen data or maze map has previously been put in zero page at locations MAPL, MAPH. The code for testing if the block's left direction is open is shown below.

```
            LDY BLOCK     ;BLOCK WE TEST IS USED AS INDEX INTO TABLE
            LDA (MAPL),Y  ;GET VALUE OF BLOCK
            AND #$01      ;TEST LEFT DIRECTION
            BEQ           ;IF RESULT 0 THEN LEAVE FLAG OPEN
            LDA #$01      ;SET FLAG CLOSED
            STA FLAG,X    ;X REG. CONTAINS PLAYER #
.2          LDA (MAPL),Y
```

Computer Controlled Players

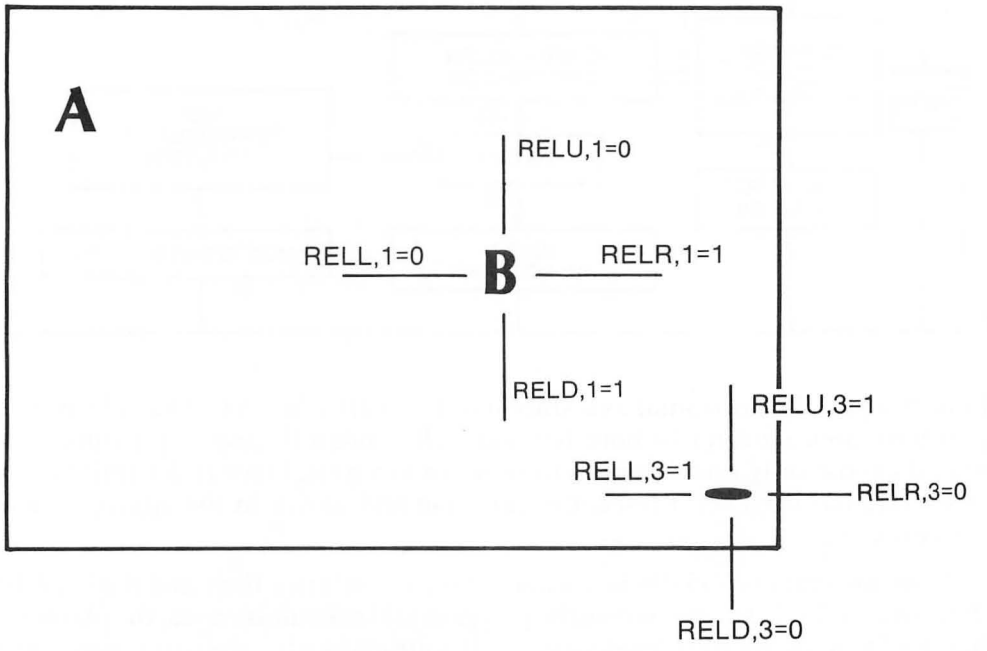
The three other players, the two letters and the minus sign, are computer controlled. The two letters must continually flee from the player's joystick controlled letter, while the minus sign homes in on the position of the player's letter.

Each of the computer controlled sprites must determine the direction or directions that they wish to travel relative to the player's letter in order to either seek it or flee from it. If the playfield were completely open the logic would be rather simple. The minus sign would approach the player in one of two random directions, horizontally or vertically until it was even in that axis, then move towards it in along the opposite axis. A letter fleeing from the player's letter would would move away from the player along one axis until it reached the playfield boundary, then move towards the corner opposite that of the player's letter. Of course it would get trapped in the corner.

Mazes have lots of corners where fleeing letters and pursuing minus signs could become trapped. A pursuing player that found itself in a parallel corridor would closely follow your motions as you moved back and forth safe but next to it. There might not even be an escape since you and it would reach the next turn at exactly the same time. On the other hand, a player that fled would often get stuck in a corner awaiting for you to become parallel with it along one corridor before fleeing along the other. As many maze game designers will testify, the only practical solution is to force both the pursueing and fleeing computer controlled players to always move forward in the corridors, never in reverse.

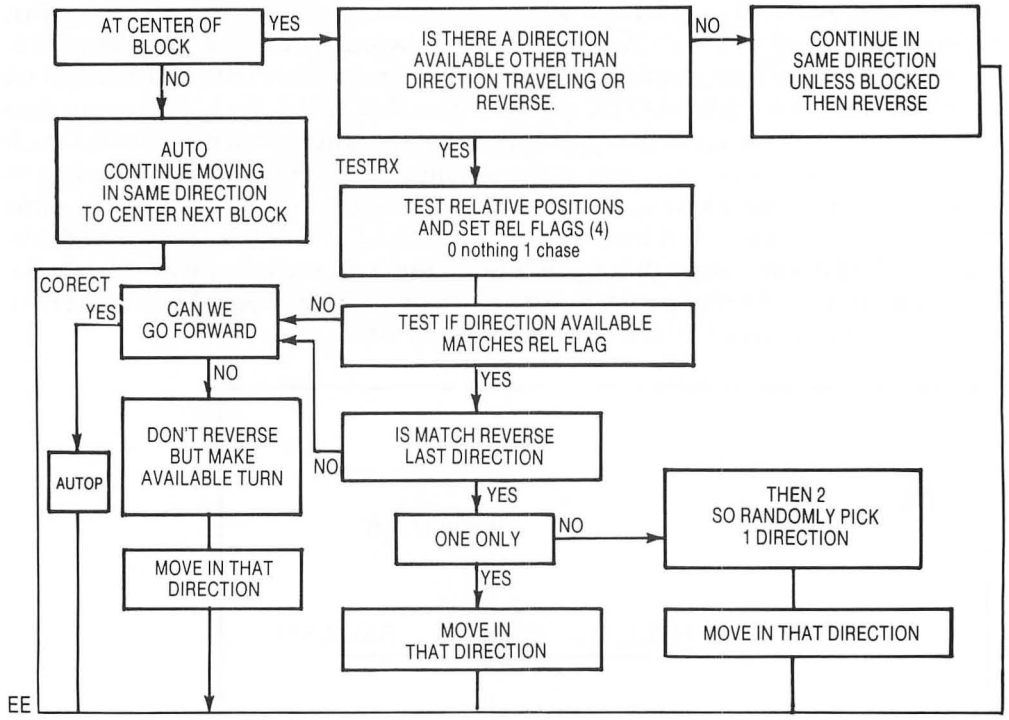
The logic needed to keep a player moving forward is not always simple, for sometimes it disobeys the general rules given to each type of player to either pursue or flee from the joystick controlled letter. In general, if a player is at a decision making point, the center of a block, it must decide if there is a direction available other than the direction it is traveling in or reverse. If it doesn't have a choice it just continues in its current direction. But if there is a choice it must decide if one of those directions is a better choice than the direction it is currently travelling.

A set of four relative direction flags, RELL,X ,RELR,X ,RELU,X and RELD,X , can be set to indicate whether the computer controlled letters and minus sign should move towards our letter or not. Each of their X and Y coordinates are checked against the joystick controlled letter's X,Y coordinate. Obviously if the player's letter is to the left of the minus sign, the minus sign would want to move left so RELL,X is set to 1. The minus sign is player #3 (X-register = 3) so that RELL,3 = 1. The minus sign would also like to move up so that RELU,3 =1. On the other hand, the letter B, which has a similiar relative positioning as the — sign in the diagram below, wants to escape and move in the exact opposite direction. The testing algorithm is the same for both types of players but instead of setting RELU,1=1 the letter B would like RELU,1=-1. If it wants to go the opposite direction it is much simpler to just set the flag for the opposite direction than to test for negative relative flags. Thus RELD,1=1 is the exact equivalent and the letter subsequently flees.



The next step is to decide if any of these relative direction flags match any of the legal direction movement flags. For example if RELL,1=1 and FLAGL,1=1 then either the fleeing letters or the pursuing minus sign can take the turn. If the move is possible it sets a move flag MFLAGL,1=1 for that direction. The reader at this point is probably muttering, "Not another flag!" It is actually necessary because there can be more than one possible direction to move. In that case the program will have to choose one direction to move. There is a counter called NUM that increments each time it sets a move flag. If NUM>1 we will have to choose a direction randomly. The choice of directions are arranged in pairs left-right and up-down. Depending on the random number value, the test will be on either the vertical or horiziontal direction first. There is no danger that the program will miss finding a set move flag if it

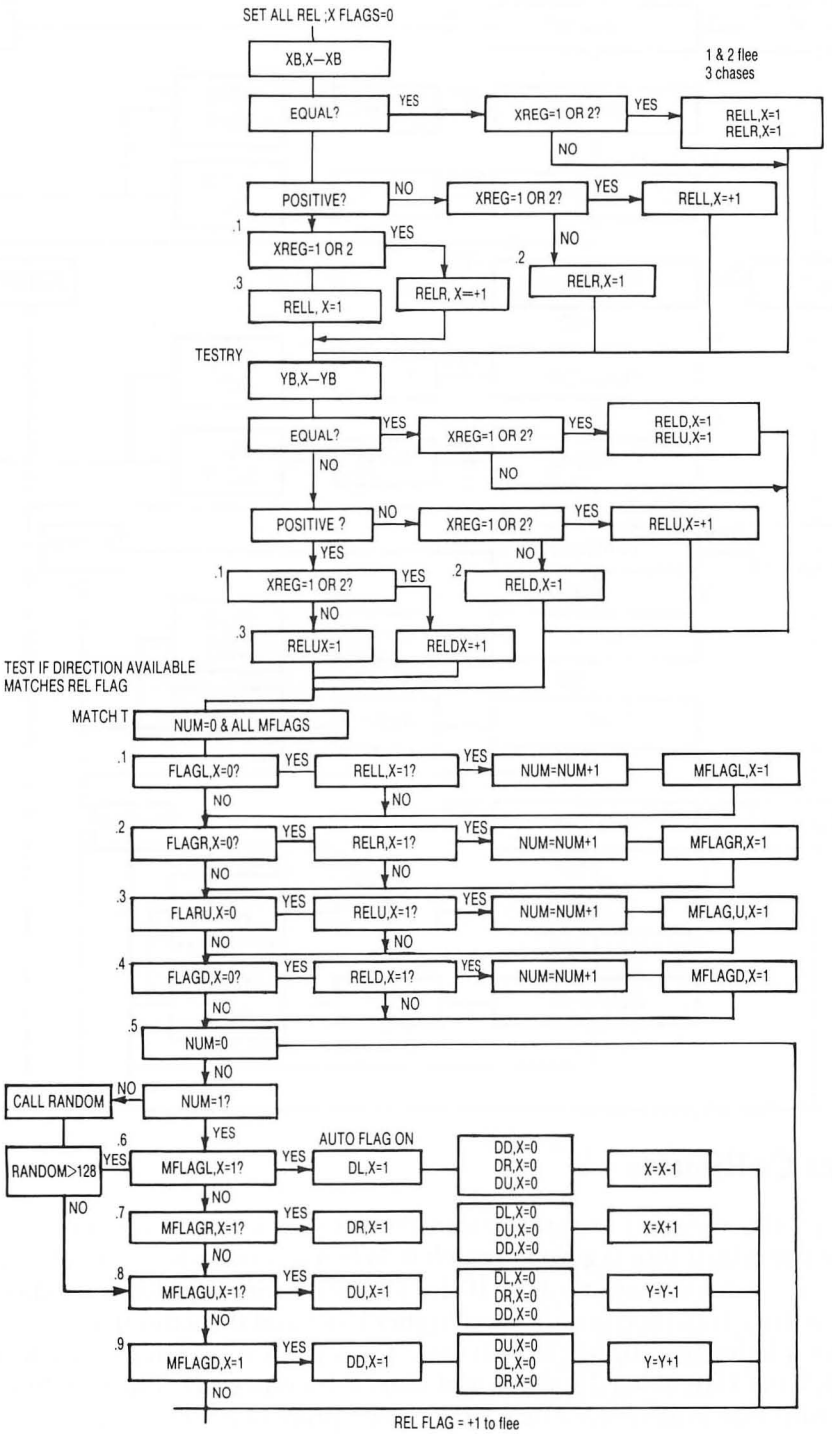
MAZE GAME (OTHER 3 PLAYERS)

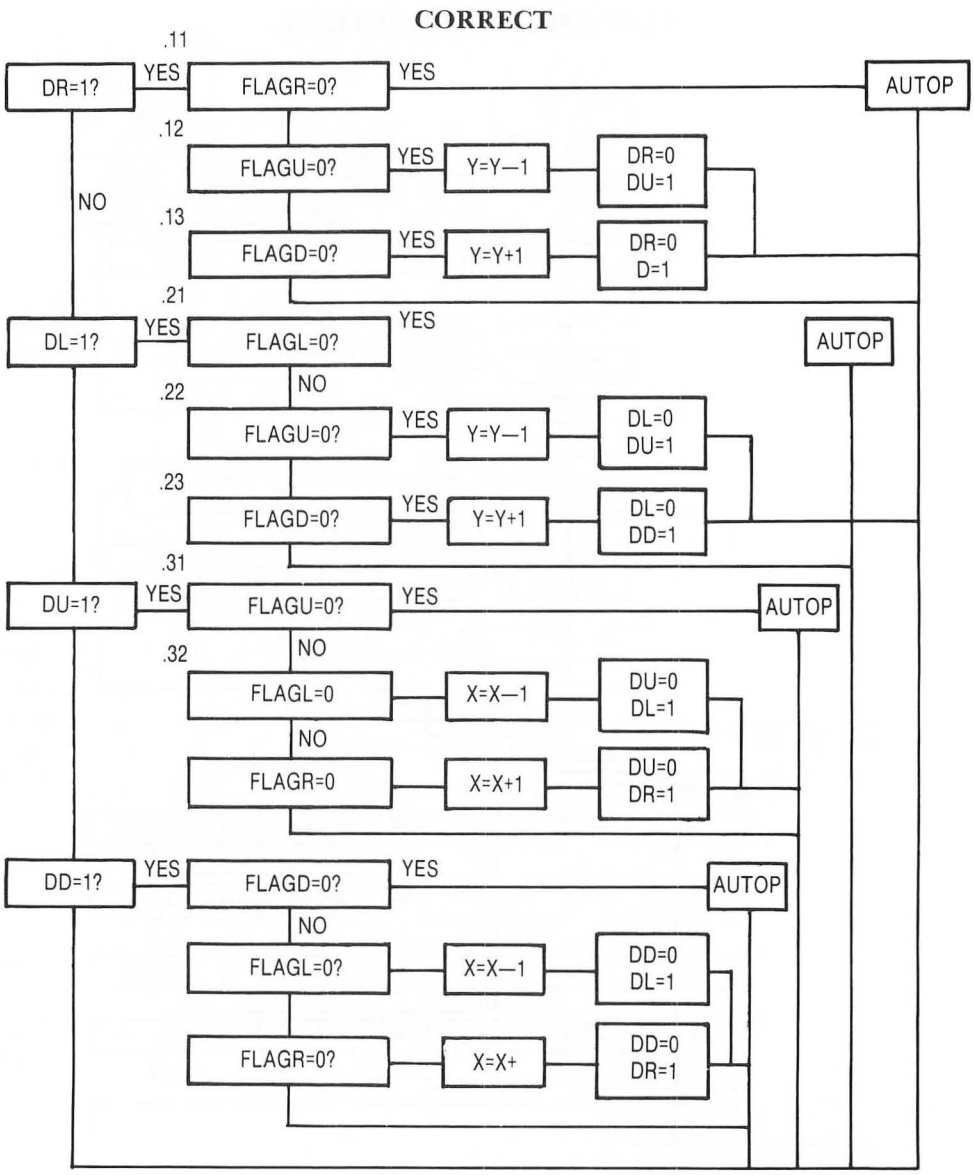


branches past the horizontal axis since it only occurs when NUM =2. The two flags that have been set can't be both left and right since a fleeing or pursuing player would choose only one direction to move in any axis. Once it determines which move flag has been set it resets the auto flag and moves in the appropriate new direction.

There are cases especially in corners where the relative flags and the legal flags don't match. In these cases where the player would become trapped, the player must be forced to make the only legal turn even if it means moving the wrong way towards the joystick controlled letter. I called the subroutine CORECT. The routine, realizing that the player was moving forward when it became stuck, tests which auto flag is set and compares it with the legal move flag for that direction. If it can continue moving forward it goes to AUTOP, the automatic mode for the player, and bypasses the code to force it to make the corner against its will. However, if it becomes stuck, it, it tests the legal move flags for the two directions along the opposite axis. For example if it were heading right and became stuck it would check FLAGU,X and FLAGD,X. There is no need to check the FLAGL,X because the player is not allowed to reverse itself. Once we have determined the new direction we reset the auto flags and move it one pixel in that direction.

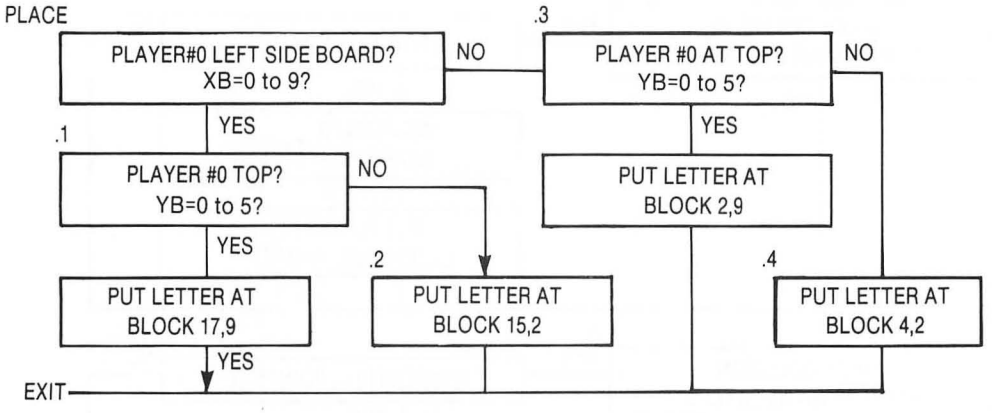
TEST & SET RELATIVE FLAGS





Player Collisions

Eventually, either the joystick controlled letter is going to catch the next higher letter or the minus sign is going to catch it. When any two players overlap a collision register is set which we can test. If the joystick controlled letter catches the next higher letter it must become the next higher letter and the letter that was just caught becomes a letter two higher than it was. Thus letter A which catches B becomes a letter B, letter C remains the same, and letter B becomes a D. The caught letter must be repositioned somewhere else on the screen preferably at the opposite end of the

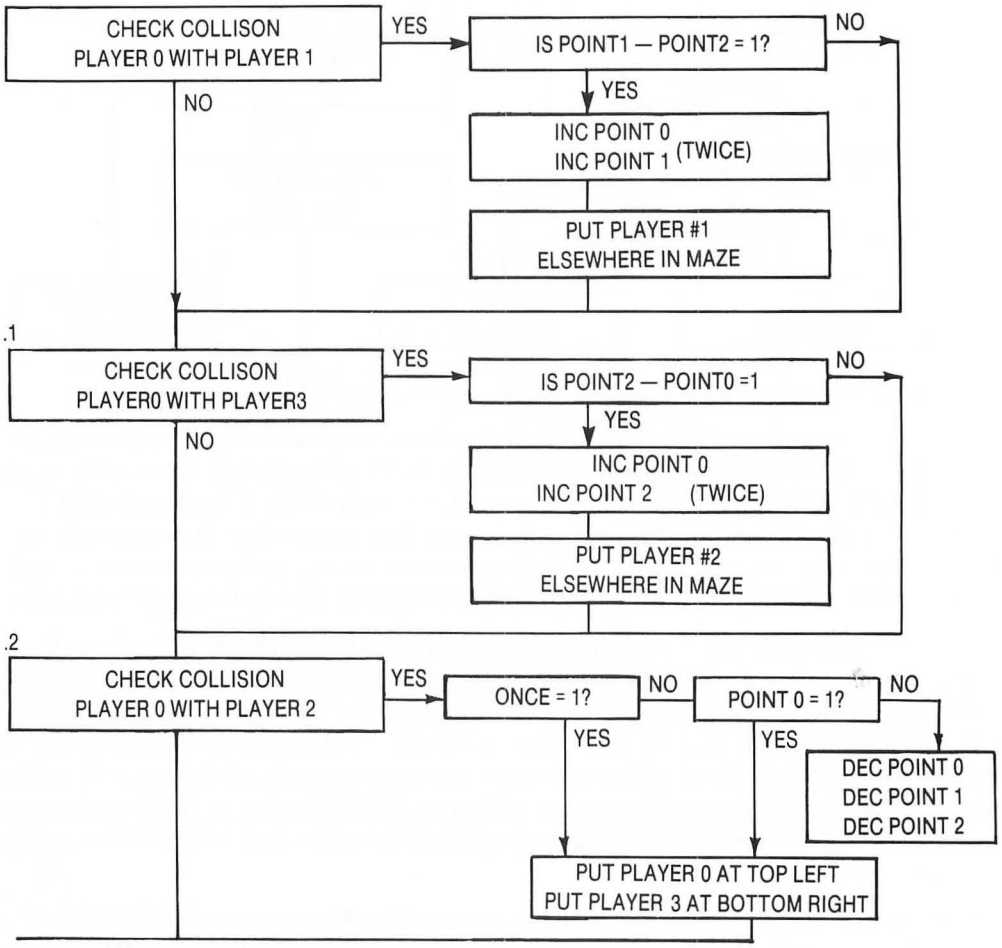


screen. If our hero is at the bottom left of the maze, we put the new letter at the top right. The four possible placement positions aren't random but in specific spots with specific starting directions. The actual repositioning is accomplished in a subroutine called PLACE. It is very simple and clearcut routine. It does randomize the left-right starting positions for the bottom two locations because occasionally two letters are caught almost simultaneously in either of the top two outer pathways.

A collision of your letter with the minus sign results in a decrement of one letter except when you are the letter A. In fact all three letters are decremented one letter so that if you are the letter C chasing a D, after the collision with the minus sign you are a B chasing a C in exactly the same position. Although the minus sign is repositioned at the bottom of the screen on the opposite side of the maze from the player, it was felt that penalizing your player by repositioning it at the top corner was frustrating to beginners who got caught all too often and felt that it was like starting over.

Each of the three lettered players has a variable that points to which letter it currently is. POINT0 refers to the joystick controlled letter, and POINT1 and POINT2 refer to the two fleeing letters. This variable also controls which letter shape is taken from the shape table during the PLOTSET subroutine. The shapes are arranged in numerical order. Shape #0 is the minus sign. The 26 letters follow in sequential order. Two blanks are placed at the end so that when the player reaches Y it is chasing only a Z and one invisible player. When the player reaches Z there are two invisible players still on the screen.

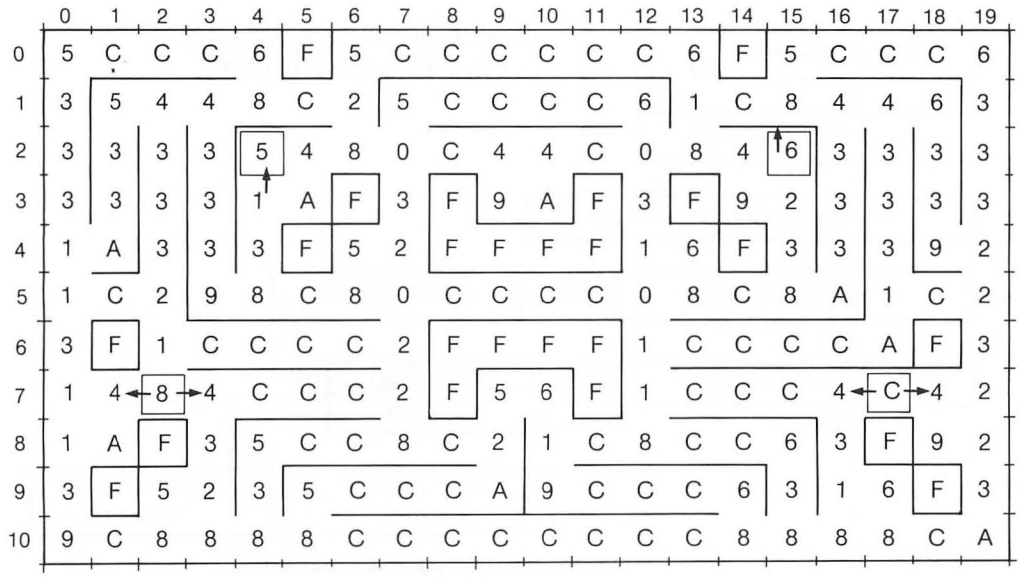
While it is possible to test whether player #1 or player #2 has collided with our joystick controlled letter, you can't be sure that the collision is with the next higher letter because that letter alternates between the two players. Therefore a test comparing the values of the two colliding letters must also be done to determine if the two letters are one apart. Say our player is an E (POINT0=5) and it collides mistakenly with player #1 that currently is a G (POINT1=7). The test POINT1 -POINT0 gives a value greater than 1. Therefore we ignore the collision. But if we collide with player #2 that is an F (POINT2=6) then POINT2–POINT0 is equal to 1. We then increment POINT0 our player to a F and increment POINT2 twice to H. Now we have a F chasing a G and H.



Second Maze Level

Eventually the player reaches the letter Z in the game. There is a four second pause before a new maze appears. Setting up the screen is simple. The 220 bytes of character data for the second screen is moved into screen memory starting at location SCREEN. The zero page pointers MAPL and MAPH, which are used by the subroutine LEGAL to obtain the value of a particular block, are also set to point to the character data at DSCREEN2.

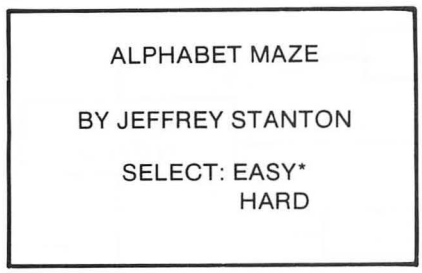
```
          LDX #$00
SLOOP1    LDA DSCREEN2,X ;LOAD NEW MAZE DATA
          STA SCREEN,X   ;STORE ON SCREEN
          CPX #$F0       ;DONE?
          BNE SLOOP1     ;NEXT BLOCK
          LDA #DSCREEN2  ;SETUP ZERO PAGE POINTERS TO DATA
          STA MAPL
          LDA /DSCREEN2
          STA MAPH
```

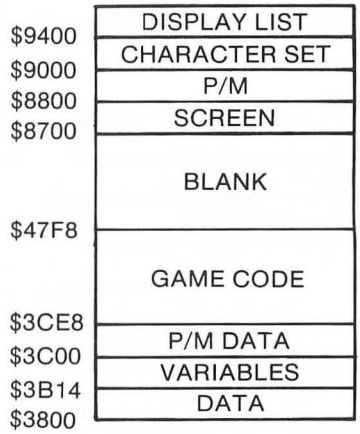


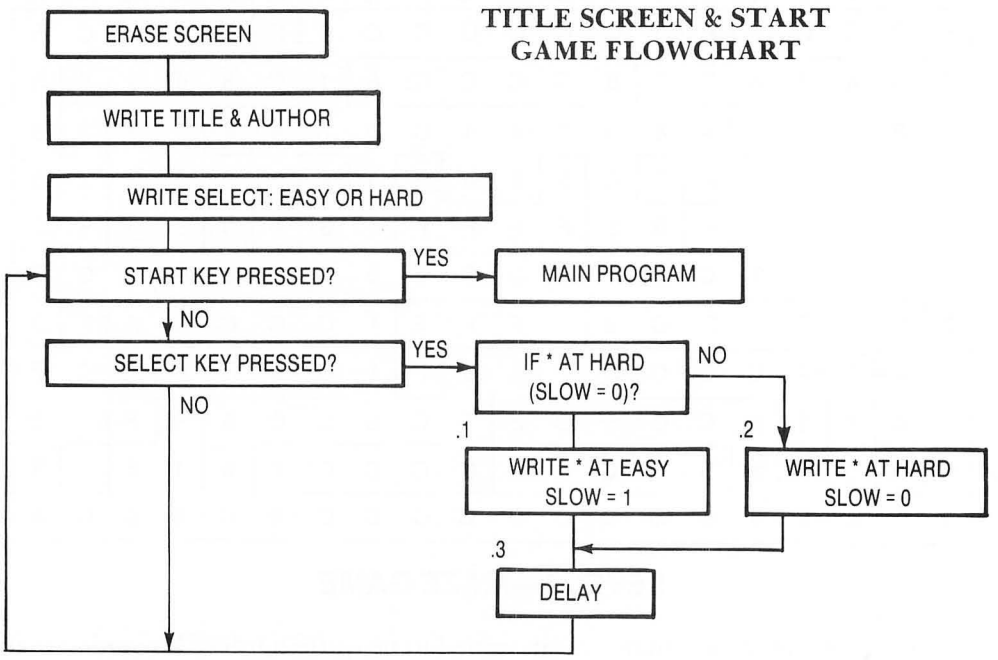
LEVEL 2—MAZE GAME

The layout of the second maze was designed to be quite harder. The only open pathways are along the bottom and two sides. The interior pathways are designed so that if you don't concentrate and choose your pathways with care, a wrong turn will nearly always give the pursued letter a bigger lead.

It is possible to add many more levels to the game. If there is only a few you could duplicate the above code and by testing which level you have just finished branch to the appropriate block of code to put the maze character data in screen memory. More advanced programmers should set up an indexed table for the hi byte pointer to each 256 byte block of character maze data if there are more than four maze levels.

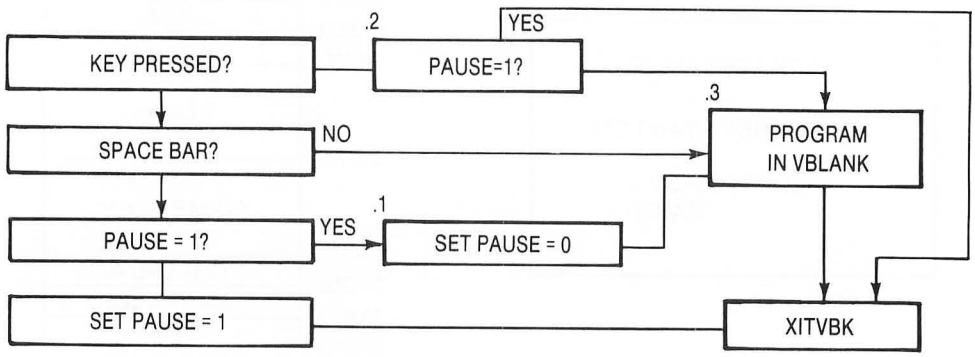






Pause Feature

A pause game feature is important to most arcade games, especially one in which the game might last more than several minutes. The problem with an Atari is that game code within the vertical blank executes 60 times per second regardless of any pause control in the main program loop. If the game is to stop, the pause code must branch past the game code within vertical blank to XITVBK. In our case, if we branch past the VBLANK flag which we use to syncronize the non-vertical blank code, the rest of the game action will also stop. The pause code shown in the flow chart below is placed at the very beginning of vertical blank.



```
                    00010 *ALPHABET MAZE - COPYRIGHT 1984 - BY JEFFREY STANTON
                    00015          .OR $3800
                    00017          .TF "D:MAZE.OBJ"
                    00020 *ZERO PAGE EQUATES
00F0:               00025 SHPL     .EQ $F0
00F1:               00030 SHPH     .EQ $F1
00F2:               00035 SHPML    .EQ $F2
00F3:               00040 SHPMH    .EQ $F3
00F4:               00045 SHPMOL   .EQ $F4
00F5:               00050 SHPMOH   .EQ $F5
00F6:               00055 PMADR    .EQ $F6
00F8:               00060 MAPL     .EQ $F8
00F9:               00065 MAPH     .EQ $F9
                    00070 *OTHER EQUATES
8700:               00075 SCREEN   .EQ $8700    ;ADR OF SCREEN
9400:               00080 NDLIST   .EQ $9400    ;ADR OF DISPLAY LIST
9000:               00085 CHRSET   .EQ $9000    ;ADR OF CHARACTER SET
0400:               00090 SETSIZ   .EQ 1024
E45C:               00095 SETVBK   .EQ $E45C
E462:               00100 XITVBK   .EQ $E462
0278:               00105 STICK    .EQ $278
02F4:               00107 CHBAS    .EQ $2F4     ;CHARACTER SET BASE
02C8:               00108 COLOR4   .EQ $2C8
                    00110 *PLAYER MISSILE EQUATES
D407:               00115 PMBASE   .EQ $D407
8800:               00120 PDATA    .EQ $8800    ;ADR OF P/M AREA
D01D:               00125 GRACTL   .EQ $D01D
022F:               00130 DMACTL   .EQ $22F
D01E:               00135 HITCLR   .EQ $D01E
D008:               00140 SIZEP0   .EQ $D008    ;PLAYER SIZES
D009:               00145 SIZEP1   .EQ $D009
D00A:               00150 SIZEP2   .EQ $D00A
D00B:               00155 SIZEP3   .EQ $D00B
02C0:               00160 COLPM0   .EQ $2C0     ;PLAYER COLORS
02C1:               00165 COLPM1   .EQ $2C1
02C2:               00170 COLPM2   .EQ $2C2
02C3:               00175 COLPM3   .EQ $2C3
D000:               00180 HPOSP0   .EQ $D000    ;HORIZ PLAYER POSITIONS
D001:               00185 HPOSP1   .EQ $D001
D002:               00190 HPOSP2   .EQ $D002
D003:               00192 HPOSP3   .EQ $D003
D00C:               00193 POPL     .EQ $D00C    ;PLAYER TO PLAYER COLLISIONS
D20A:               00195 RANDOM   .EQ $D20A
D01F:               00205 CONSOL   .EQ $D01F
                    00210 *
3800: 05 0C 0C
3803: 0C 0C 0C
3806: 0C 0C 06
3809: 0F            00215 DSCREEN  .HS 050C0C0C0C0C0C0C060F
380A: 0F 05 0C
380D: 0C 0C 0C
3810: 0C 0C 0C
3813: 06            00220          .HS 0F050C0C0C0C0C0C0C06
3814: 03 05 0C
3817: 0C 04 0C
381A: 0C 0C 00
381D: 0C            00225          .HS 03050C0C040C0C0C000C
381E: 0C 00 0C
3821: 0C 0C 04
3824: 0C 0C 06
3827: 03            00230          .HS 0C000C0C0C040C0C0603
3828: 03 03 0F
```

```
382B: 0F 03 05
382E: 0C 0C 02
3831: 0F        00235          .HS 03030F0F03050C0C020F
3832: 0F 01 0C
3835: 0C 06 03
3838: 0F 0F 03
383B: 03        00240          .HS 0F010C0C06030F0F0303
383C: 01 0A 0F
383F: 0F 09 02
3842: 0F 0F 01
3845: 0C        00245          .HS 010A0F0F09020F0F010C
3846: 0C 02 0F
3849: 0F 01 0A
384C: 0F 0F 09
384F: 02        00250          .HS 0C020F0F010A0F0F0902
3850: 01 0C 0C
3853: 0C 0C 02
3856: 0F 05 0A
3859: 0F        00255          .HS 010C0C0C0C020F050A0F
385A: 0F 09 06
385D: 0F 01 0C
3860: 0C 0C 0C
3863: 02        00260          .HS 0F09060F010C0C0C0C02
3864: 03 0F 0F
3867: 0F 0F 01
386A: 0C 02 0F
386D: 0F        00265          .HS 030F0F0F0F010C020F0F
386E: 0F 0F 01
3871: 0C 02 0F
3874: 0F 0F 0F
3877: 03        00270          .HS 0F0F010C020F0F0F0F03
3878: 03 05 0C
387B: 0C 0C 02
387E: 0F 09 06
3881: 0F        00275          .HS 03050C0C0C020F09060F
3882: 0F 05 0A
3885: 0F 01 0C
3888: 0C 0C 06
388B: 03        00280          .HS 0F050A0F010C0C0C0603
388C: 03 01 0C
388F: 0C 06 03
3892: 0F 0F 01
3895: 04        00285          .HS 03010C0C06030F0F0104
3896: 04 02 0F
3899: 0F 03 05
389C: 0C 0C 02
389F: 03        00290          .HS 04020F0F03050C0C0203
38A0: 03 03 0F
38A3: 0F 03 09
38A6: 0C 0C 02
38A9: 03        00295          .HS 03030F0F03090C0C0203
38AA: 03 01 0C
38AD: 0C 0A 03
38B0: 0F 0F 03
38B3: 03        00300          .HS 03010C0C0A030F0F0303
38B4: 03 09 0C
38B7: 06 09 0C
38BA: 0C 0C 0A
38BD: 03        00305          .HS 03090C06090C0C0C0A03
38BE: 03 09 0C
38C1: 0C 0C 0A
38C4: 05 0C 0A
```

```
38C7: 03       00310          .HS 03090C0C0C0A050C0A03
38C8: 09 0C 0C
38CB: 08 0C 0C
38CE: 0C 0C 0C
38D1: 0A       00315          .HS 090C0C080C0C0C0C0C0A
38D2: 09 0C 0C
38D5: 0C 0C 0C
38D8: 08 0C 0C
38DB: 0A       00320          .HS 090C0C0C0C0C080C0C0A
38DC: 00 00 00
38DF: 00 00 00
38E2: 00 00 00
38E5: 00       00325          .HS 00000000000000000000
38E6: 00 00 00
38E9: 00 00 00
38EC: 00 00 00
38EF: 00       00330          .HS 00000000000000000000
38F0:          00335          .BS $10
3900: 05 0C 0C
3903: 0C 06 0F
3906: 05 0C 0C
3909: 0C       00340 DSCREEN2 .HS 050C0C0C060F050C0C0C
390A: 0C 0C 0C
390D: 06 0F 05
3910: 0C 0C 0C
3913: 06       00345          .HS 0C0C0C060F050C0C0C06
3914: 01 04 04
3917: 04 08 0C
391A: 02 05 0C
391D: 0C       00350          .HS 01040404080C02050C0C
391E: 0C 0C 06
3921: 01 0C 08
3924: 04 04 04
3927: 02       00355          .HS 0C0C06010C0804040402
3928: 03 03 03
392B: 03 05 04
392E: 08 00 0C
3931: 04       00360          .HS 03030303050408000C04
3932: 04 0C 00
3935: 08 04 06
3938: 03 03 03
393B: 03       00365          .HS 040C00080406030303303
393C: 03 03 03
393F: 03 01 0A
3942: 0F 03 0F
3945: 09       00370          .HS 03030303010A0F030F09
3946: 0A 0F 03
3949: 0F 09 02
394C: 03 03 03
394F: 03       00375          .HS 0A0F030F090203030303
3950: 01 0A 03
3953: 03 03 0F
3956: 05 02 0F
3959: 0F       00380          .HS 010A0303030F05020F0F
395A: 0F 0F 01
395D: 06 0F 03
3960: 03 03 09
3963: 02       00385          .HS 0F0F01060F0303030902
3964: 01 0C 02
3967: 09 08 0C
396A: 08 00 0C
396D: 0C       00390          .HS 010C0209080C08000C0C
```

```
396E: 0C 0C 00
3971: 08 0C 08
3974: 0A 01 0C
3977: 02        00395           .HS 0C0C00080C080A010C02
3978: 03 0F 01
397B: 0C 0C 0C
397E: 0C 02 0F
3981: 0F        00400           .HS 030F010C0C0C0C020F0F
3982: 0F 0F 01
3985: 0C 0C 0C
3988: 0C 0A 0F
398B: 03        00405           .HS 0F0F010C0C0C0C0A0F03
398C: 01 04 08
398F: 04 0C 0C
3992: 0C 02 0F
3995: 05        00410           .HS 010408040C0C0C020F05
3996: 06 0F 01
3999: 0C 0C 0C
399C: 04 0C 04
399F: 02        00415           .HS 060F010C0C0C040C0402
39A0: 01 0A 0F
39A3: 03 05 0C
39A6: 0C 08 0C
39A9: 02        00420           .HS 010A0F03050C0C080C02
39AA: 01 0C 08
39AD: 0C 0C 06
39B0: 03 0F 09
39B3: 02        00425           .HS 010C080C0C06030F0902
39B4: 03 0F 05
39B7: 02 03 05
39BA: 0C 0C 0C
39BD: 0A        00430           .HS 030F050203050C0C0C0A
39BE: 09 0C 0C
39C1: 0C 06 03
39C4: 01 06 0F
39C7: 03        00435           .HS 090C0C0C060301060F03
39C8: 09 0C 08
39CB: 08 08 08
39CE: 0C 0C 0C
39D1: 0C        00440           .HS 090C080808080C0C0C0C
39D2: 0C 0C 0C
39D5: 0C 08 08
39D8: 08 08 0C
39DB: 0A        00445           .HS 0C0C0C0C080808080C0A
39DC: 00 00 00
39DF: 00 00 00
39E2: 00 00 00
39E5: 00        00450           .HS 00000000000000000000
39E6: 00 00 00
39E9: 00 00 00
39EC: 00 00 00
39EF: 00        00455           .HS 00000000000000000000
39F0:           00460           .BS $10
3A00: 70 70 70
3A03: 47 00 87
3A06: 07 07     00465 DLIST     .HS 7070704700870707
3A08: 07 07 07
3A0B: 07 07 07
3A0E: 07 07     00470           .HS 0707070707070707
3A10: 07 41 00
3A13: 94        00475           .HS 07410094
                00480 *MAP CHARACTER DATA
```

```
3A14: 00 00 00
3A17: 00 00 00
3A1A: 00 00     00485 DCHAR     .HS 0000000000000000
3A1C: 80 80 80
3A1F: 80 80 80
3A22: 80 80     00490           .HS 8080808080808080
3A24: 01 01 01
3A27: 01 01 01
3A2A: 01 01     00495           .HS 0101010101010101
3A2C: 81 81 81
3A2F: 81 81 81
3A32: 81 81     00500           .HS 8181818181818181
3A34: FF 00 00
3A37: 00 00 00
3A3A: 00 00     00505           .HS FF00000000000000
3A3C: FF 80 80
3A3F: 80 80 80
3A42: 80 80     00510           .HS FF80808080808080
3A44: FF 01 01
3A47: 01 01 01
3A4A: 01 01     00515           .HS FF01010101010101
3A4C: FF 81 81
3A4F: 81 81 81
3A52: 81 81     00520           .HS FF81818181818181
3A54: 00 00 00
3A57: 00 00 00
3A5A: 00 FF     00525           .HS 00000000000000FF
3A5C: 80 80 80
3A5F: 80 80 80
3A62: 80 FF     00530           .HS 80808080808080FF
3A64: 01 01 01
3A67: 01 01 01
3A6A: 01 FF     00535           .HS 01010101010101FF
3A6C: 81 81 81
3A6F: 81 81 81
3A72: 81 FF     00540           .HS 81818181818181FF
3A74: FF 00 00
3A77: 00 00 00
3A7A: 00 FF     00545           .HS FF000000000000FF
3A7C: FF 80 80
3A7F: 80 80 80
3A82: 80 FF     00550           .HS FF808080808080FF
3A84: FF 01 01
3A87: 01 01 01
3A8A: 01 FF     00555           .HS FF010101010101FF
3A8C: FF FF FF
3A8F: FF FF FF
3A92: FF FF     00560           .HS FFFFFFFFFFFFFFFF
3A94: 20 20 20
3A97: 41 4C 50
3A9A: 48 41 42
3A9D: 45        00565 TITLE     .HS 202020414C5048414245
3A9E: 54 20 4D
3AA1: 41 5A 45
3AA4: 20 20 20
3AA7: 20        00566           .HS 54204D415A4520202020
3AA8: 20 20 20
3AAB: 20 20 20
3AAE: 20 20 20
3AB1: 20        00570           .HS 20202020202020202020
3AB2: 20 20 20
3AB5: 20 20 20
```

```
3AB8: 20 20 20
3ABB: 20        00571           .HS 20202020202020202020
3ABC: 20 42 59
3ABF: 20 4A 45
3AC2: 46 46 52
3AC5: 45        00575           .HS 204259204A4546465245
3AC6: 59 20 53
3AC9: 54 41 4E
3ACC: 54 4F 4E
3ACF: 20        00576           .HS 59205354414E544F4E20
3AD0: 00 00 00
3AD3: 33 25 2C
3AD6: 25 23 34
3AD9: 1A 00 25
3ADC: 21 33 39
3ADF: 00 00 00
3AE2: 00 00     00580 TITLE1    .AT '   SELECT: EASY     '
3AE4: 00 00 00
3AE7: 00 00 00
3AEA: 00 00 00
3AED: 00 00 28
3AF0: 21 32 24
3AF3: 00 00 00
3AF6: 00 00     00585           .AT '           HARD     '
3AF8:           00590           .BS $08
3B00: 00 00 00
3B03: 00 00 2E
3B06: 25 38 34
3B09: 00 2C 25
3B0C: 36 25 2C
3B0F: 00 00 00
3B12: 00 00     00595 TITLE2    .AT '     NEXT LEVEL     '
                00600 *VARIABLES
3B14:           00605 XO        .BS 1       ;PLAYER'S MAN POSITION
3B15:           00610 X         .BS 4       ;REST OF PLAYER POSITIONS
3B19:           00615 YO        .BS 1
3B1A:           00620 Y         .BS 4       ;REST OF PLAYER POSITIONS
3B1E:           00625 XB        .BS 4       ;BLOCK EACH PLAYER IN
3B22:           00630 YB        .BS 4
3B26:           00635 FLAGL     .BS 4       ;LEGAL MOVE FLAGS FOR EACH PLAYER
3B2A:           00640 FLAGR     .BS 4
3B2E:           00645 FLAGU     .BS 4
3B32:           00650 FLAGD     .BS 4
3B36:           00655 RELL      .BS 4       ;DIRECTION WANT TO MOVE IN FLAGS
3B3A:           00660 RELR      .BS 4
3B3E:           00665 RELU      .BS 4
3B42:           00670 RELD      .BS 4
3B46:           00675 MFLAGL    .BS 4       ;MATCH FLAGS
3B4A:           00680 MFLAGR    .BS 4
3B4E:           00685 MFLAGU    .BS 4
3B52:           00690 MFLAGD    .BS 4
3B56:           00695 TEMPX     .BS 1
3B57:           00700 TEMPY     .BS 1
3B58:           00705 TEMPL     .BS 4
3B5C:           00710 TEMPH     .BS 4
3B60:           00715 BLOCK     .BS 1       ;OFFSET IN SCREEN MEMORY OF CURRENT BLOCK
3B61: 01        00720 POINTO    .DA #1      ;SHAPE # PLAYER O
3B62: 02        00725 POINT1    .DA #2
3B63: 03        00730 POINT2    .DA #3
3B64: 00        00735 POINT3    .DA #0
3B65:           00740 DL        .BS 4       ;AUTO FLAGS
3B69:           00745 DR        .BS 4
```

```
3B6D:          00750 DU       .BS 4
3B71:          00755 DD       .BS 4
3B75: 00       00820 INHIBIT  .DA #0      ;PREVENTS OBTAINING TWO JOYSTICK DIRECTIONS
3B76: 00       00825 VBFLAG   .DA #0      ;VBLANK FINISHED FLAG   \ON DIAGONAL
3B77: 00       00830 LFLAG    .DA #0      ;INDICATES WHETHER AT CENTER OF BLOCK
3B78: 00       00835 NUM      .DA #0      ;NUMBER OF DIRECTION MATCH FLAGS
3B79: 00       00840 PAUSE    .DA #0      ;PAUSE FLAG
3B7A: 00       00845 SLOW     .DA #0      ;SETS MINUS SIGN TO HALF SPEED
3B7B: 00       00850 HALF     .DA #0      ;COUNTER WHEN PLAYER MOVES AT HALF SPEED
3B7C:          00855 LEVEL    .BS 1       ;MAZE LEVEL
3B7D:          00857 ONCE     .BS 1       ;PREVENTS DOUBLE COLLISIONS @ HALF SPEED
               00860 *POINTERS TO PM DATA
3B7E: 00 08 10
3B81: 18 20 28
3B84: 30 38    00865 SHPLO    .HS 0008101820283038
3B86: 40 48 50
3B89: 58 60 68
3B8C: 70 78    00870          .HS 4048505860687078
3B8E: 80 88 90
3B91: 98 A0 A8
3B94: B0 B8    00875          .HS 80889098A0A8B0B8
3B96: C0 C8 D0
3B99: D8 E0    00880          .HS C0C8D0D8E0
3B9B:          00885          .BS $65
               00890 *MAKE SURE STARTS @$2C00
               00895 *PLAYER MISSILE DATA
3C00: 00 00 00
3C03: 3C 3C 00
3C06: 00 00    00900 SHAPES   .HS 0000003C3C000000 ;MINUS SIGN
3C08: 18 3C 66
3C0B: 42 7E 42
3C0E: 42 42    00905          .HS 183C66427E424242 ;LETTER A
3C10: 7C 42 42
3C13: 7C 42 42
3C16: 42 7C    00910          .HS 7C42427C4242427C ;B
3C18: 1E 20 40
3C1B: 40 40 40
3C1E: 20 1E    00915          .HS 1E2040404040201E ;C
3C20: 78 44 42
3C23: 42 42 42
3C26: 44 78    00920          .HS 7844424242424478 ;D
3C28: 7E 40 40
3C2B: 7C 40 40
3C2E: 40 7E    00925          .HS 7E40407C4040407E ;E
3C30: 7E 40 40
3C33: 7C 40 40
3C36: 40 40    00930          .HS 7E40407C40404040 ;F
3C38: 1E 20 40
3C3B: 40 4E 42
3C3E: 22 1C    00935          .HS 1E2040404E42221C ;G
3C40: 42 42 42
3C43: 7E 42 42
3C46: 42 42    00940          .HS 4242427E42424242 ;H
3C48: 7C 10 10
3C4B: 10 10 10
3C4E: 10 7C    00945          .HS 7C1010101010107C ;I
3C50: 04 04 04
3C53: 04 04 44
3C56: 44 38    00950          .HS 0404040404444438 ;J
3C58: 44 48 50
3C5B: 60 60 50
3C5E: 48 44    00955          .HS 4448506060504844 ;K
```

```
3C60: 40 40 40
3C63: 40 40 40
3C66: 40 7E     00960           .HS 404040404040407E ;L
3C68: 41 63 55
3C6B: 49 41 41
3C6E: 41 41     00965           .HS 4163554941414141 ;M
3C70: 62 62 52
3C73: 52 4A 4A
3C76: 46 46     00970           .HS 626252524A4A4646 ;N
3C78: 18 24 42
3C7B: 42 42 42
3C7E: 24 18     00975           .HS 1824424242422418 ;O
3C80: 7C 42 42
3C83: 42 7C 40
3C86: 40 40     00980           .HS 7C4242427C404040 ;P
3C88: 18 24 42
3C8B: 42 42 42
3C8E: 26 1B     00985           .HS 182442424242261B ;Q
3C90: 7C 42 42
3C93: 42 7C 44
3C96: 42 41     00990           .HS 7C4242427C444241 ;R
3C98: 3E 40 40
3C9B: 40 3C 02
3C9E: 02 7C     00995           .HS 3E4040403C02027C ;S
3CA0: FE 10 10
3CA3: 10 10 10
3CA6: 10 10     01000           .HS FE10101010101010 ;T
3CA8: 42 42 42
3CAB: 42 42 42
3CAE: 42 3C     01005           .HS 424242424242423C ;U
3CB0: 82 82 82
3CB3: 82 44 44
3CB6: 28 10     01010           .HS 8282828244442810 ;V
3CB8: 41 41 41
3CBB: 41 49 55
3CBE: 63 41     01015           .HS 4141414149556341 ;W
3CC0: 41 22 14
3CC3: 08 14 22
3CC6: 41 00     01020           .HS 4122140814224100 ;X
3CC8: 42 42 24
3CCB: 18 10 20
3CCE: 40 40     01025           .HS 4242241810204040 ;Y
3CD0: 7F 02 04
3CD3: 08 10 20
3CD6: 7F 00     01030           .HS 7F02040810207F00 ;Z
3CD8: 00 00 00
3CDB: 00 00 00
3CDE: 00 00     01035           .HS 0000000000000000 ;BLANK
3CE0: 00 00 00
3CE3: 00 00 00
3CE6: 00 00     01040           .HS 0000000000000000 ;BLANK
                01045 *SETUP DLIST
                01050 BEGIN
3CE8: A2 00     01055           LDX #$00
3CEA: BD 00 3A 01060 DLOOP      LDA DLIST,X
3CED: 9D 00 94 01065            STA NDLIST,X
3CF0: E8       01070            INX
3CF1: E0 15    01075            CPX #$15 21 ELEMENTS
3CF3: D0 F5    01080            BNE DLOOP
3CF5: A9 00    01085            LDA #NDLIST ;LOCATION OF DISPLAY LIST
3CF7: 8D 30 02 01090            STA $230
3CFA: A9 94    01095            LDA /NDLIST
```

```
3CFC: 8D 31 02 01100          STA $231
              01105 *ERASE SCREEN
3CFF: A2 00    01110 START    LDX #$00
3D01: A9 00    01115          LDA #$00
3D03: 9D 00 87 01120 ALOOP    STA SCREEN,X
3D06: E8       01125          INX
3D07: D0 FA    01130          BNE ALOOP
              01135 *WRITE TITLE & AUTHOR
3D09: A9 E0    01140          LDA #$E0    ;ROM CHARACTER SET
3D0B: 8D F4 02 01145          STA CHBAS
3D0E: A2 00    01150          LDX #$00
3D10: BD 94 3A 01155 BLOOP    LDA TITLE,X
3D13: 38       01157          SEC
3D14: E9 20    01158          SBC #$20
3D16: 9D 3C 87 01160          STA $873C,X
3D19: E8       01165          INX
3D1A: E0 3C    01170          CPX #$3C
3D1C: D0 F2    01175          BNE BLOOP
              01180 *WRITE SELECT EASY & HARD
3D1E: A2 00    01185          LDX #$00
3D20: BD D0 3A 01190 HLOOP    LDA TITLE1,X ;PUT IN 8TH ROW
3D23: 9D A0 87 01195          STA $87A0,X
3D26: E8       01200          INX
3D27: E0 28    01205          CPX #$28
3D29: D0 F5    01210          BNE HLOOP
3D2B: A9 0A    01215          LDA #$0A     ;WRITE * BESIDE HARD
3D2D: 8D C3 87 01220          STA $87C3
              01225 *READ CONSOLE KEYS
3D30: A9 00    01230          LDA #$00
3D32: 8D 7A 3B 01235          STA SLOW
3D35: 8D 7C 3B 01240          STA LEVEL
3D38: AD 1F D0 01245 KEY      LDA CONSOL
3D3B: C9 06    01250          CMP #$06
3D3D: F0 39    01255          BEQ MAIN
3D3F: C9 05    01260          CMP #$05
3D41: D0 F5    01265          BNE KEY
3D43: AD 7A 3B 01270          LDA SLOW
3D46: D0 12    01275          BNE .2
3D48: A9 0A    01280 .1       LDA #$0A    ;WRITE * BESIDE EASY
3D4A: 8D AF 87 01285          STA $87AF
3D4D: A9 00    01290          LDA #$00    ;ERASE * BESIDE HARD
3D4F: 8D C3 87 01295          STA $87C3
3D52: A9 01    01300          LDA #$01
3D54: 8D 7A 3B 01305          STA SLOW
3D57: 4C 69 3D 01310          JMP .3
3D5A: A9 0A    01315 .2       LDA #$0A    ;WRITE * BESIDE HARD
3D5C: 8D C3 87 01320          STA $87C3
3D5F: A9 00    01325          LDA #$00    ;ERASE * BESIDE EASY
3D61: 8D AF 87 01330          STA $87AF
3D64: A9 00    01335          LDA #$00
3D66: 8D 7A 3B 01340          STA SLOW
3D69: A9 E8    01345 .3       LDA #$E8
3D6B: 85 14    01350          STA $14
3D6D: A9 00    01355          LDA #$00
3D6F: 85 13    01360          STA $13
3D71: A5 13    01365 .4       LDA $13     ;DELAY 24 JIFFIES
3D73: F0 FC    01370          BEQ .4
3D75: 4C 38 3D 01375          JMP KEY
              01380 *SETUP CHARACTER SET
3D78: A2 00    01385 MAIN     LDX #$00
3D7A: BD 14 3A 01390 CLOOP    LDA DCHAR,X
3D7D: 9D 00 90 01395          STA CHRSET,X
```

```
3D80: E8       01400          INX
3D81: E0 80    01405          CPX #$80
3D83: D0 F5    01410          BNE CLOOP
3D85: A9 90    01415          LDA /CHRSET
3D87: 8D F4 02 01420          STA CHBAS
               01425 *SETUP SCREEN
3D8A: A2 00    01430          LDX #$00
3D8C: BD 00 38 01435 SLOOP    LDA DSCREEN,X
3D8F: 9D 00 87 01440          STA SCREEN,X
3D92: E8       01445          INX
3D93: E0 F0    01450          CPX #$F0    ;220 BYTES
3D95: D0 F5    01455          BNE SLOOP
3D97: A9 00    01460          LDA #DSCREEN
3D99: 85 F8    01465          STA MAPL
3D9B: A9 38    01470          LDA /DSCREEN
3D9D: 85 F9    01475          STA MAPH
3D9F: A9 00    01480          LDA #$00    ;BACKGROUND BLACK
3DA1: 8D C8 02 01485          STA COLOR4
               01490 *INITALIZE PLAYERS
3DA4: A9 88    01495          LDA #$88
3DA6: 8D 07 D4 01500          STA PMBASE
3DA9: A9 03    01505          LDA #$03
3DAB: 8D 1D D0 01510          STA GRACTL
3DAE: A9 3E    01515          LDA #$3E
3DB0: 8D 2F 02 01520          STA DMACTL
3DB3: A9 00    01525          LDA #$00
3DB5: 8D 08 D0 01530          STA SIZEP0
3DB8: 8D 09 D0 01535          STA SIZEP1
3DBB: 8D 0A D0 01540          STA SIZEP2
3DBE: 8D 0B D0 01545          STA SIZEP3
3DC1: 8D 1E D0 01550 RESTART  STA HITCLR   ;CLEAR COLLISION REG
               01555 *PLAYER #0
3DC4: A9 7A    01560          LDA #$7A     ;PLAYER #0 122 BLUE LUM10
3DC6: 8D C0 02 01565          STA COLPM0
3DC9: A9 30    01570          LDA #$30     ;INITIAL POS X=48,Y=36
3DCB: 8D 14 3B 01575          STA X0
3DCE: 8D 15 3B 01580          STA X
3DD1: 8D 00 D0 01585          STA HPOSP0   ;TELL ANTIC
3DD4: A9 24    01590          LDA #$24
3DD6: 8D 19 3B 01595          STA Y0
3DD9: 8D 1A 3B 01600          STA Y
               01605 *INITIAL OLD PLAYER #0 MEMORY POINTERS
3DDC: A9 40    01610          LDA #$40
3DDE: 8D 58 3B 01615          STA TEMPL
3DE1: A9 88    01620          LDA /PDATA
3DE3: 18       01625          CLC
3DE4: 69 04    01630          ADC #$04
3DE6: 8D 5C 3B 01635          STA TEMPH
               01640 *PLAYER #1
3DE9: A2 01    01645          LDX #$01
3DEB: A9 C8    01650          LDA #$C8     ;PLAYER #1 200 GREEN LUM8
3DED: 8D C1 02 01655          STA COLPM1
3DF0: A9 88    01660          LDA #$88     ;INITIAL POSITION X=136,Y=164
3DF2: 9D 15 3B 01665          STA X,X
3DF5: 8D 01 D0 01670          STA HPOSP1
3DF8: A9 A4    01675          LDA #$A4
3DFA: 9D 1A 3B 01680          STA Y,X
               01685 *OLD PLAYER #1 MEMORY POINTERS
3DFD: A9 40    01690          LDA #$40
3DFF: 9D 58 3B 01695          STA TEMPL,X
3E02: A9 88    01700          LDA /PDATA
3E04: 18       01705          CLC
```

```
3E05: 69 05    01710          ADC #$05
3E07: 9D 5C 3B 01715          STA TEMPH,X
               01720 *PLAYER #2
3E0A: A2 02    01725          LDX #$02
3E0C: A9 C8    01730          LDA #$C8     ;PLAYER #2 200 GREEN LUM8
3E0E: 8D C2 02 01735          STA COLPM2
3E11: A9 A0    01740          LDA #$A0     ;INITIAL POSITION X=160,Y=52
3E13: 9D 15 3B 01745          STA X,X
3E16: 8D 02 D0 01750          STA HPOSP2
3E19: A9 34    01755          LDA #$34
3E1B: 9D 1A 3B 01760          STA Y,X
               01765 *OLD PLAYER #2 MEMORY POINTERS
3E1E: A9 40    01770          LDA #$40
3E20: 9D 58 3B 01775          STA TEMPL,X
3E23: A9 88    01780          LDA /PDATA
3E25: 18       01785          CLC
3E26: 69 06    01790          ADC #$06
3E28: 9D 5C 3B 01795          STA TEMPH,X
               01800 *PLAYER #3
3E2B: A2 03    01805          LDX #$03
3E2D: A9 44    01810          LDA #$44     ;PLAYER #3 68 RED LUM4
3E2F: 8D C3 02 01815          STA COLPM3
3E32: A9 C8    01820          LDA #$C8     ;INITIAL POSITION X=200,Y=196
3E34: 9D 15 3B 01825          STA X,X
3E37: 8D 03 D0 01830          STA HPOSP3
3E3A: A9 C4    01835          LDA #$C4
3E3C: 9D 1A 3B 01840          STA Y,X
               01845 *OLD PLAYER #3 MEMORY POINTERS
3E3F: A9 40    01850          LDA #$40
3E41: 9D 58 3B 01855          STA TEMPL,X
3E44: A9 88    01860          LDA /PDATA
3E46: 18       01865          CLC
3E47: 69 07    01870          ADC #$07
3E49: 9D 5C 3B 01875          STA TEMPH,X
               01880 *INIT POINT# VALUES
3E4C: A9 01    01885          LDA #$01
3E4E: 8D 61 3B 01890          STA POINT0
3E51: A9 02    01895          LDA #$02
3E53: 8D 62 3B 01900          STA POINT1
3E56: A9 03    01905          LDA #$03
3E58: 8D 63 3B 01910          STA POINT2
3E5B: A9 00    01915          LDA #$00
3E5D: 8D 64 3B 01920          STA POINT3
3E60: 8D 7D 3B 01922          STA ONCE
               01925 *INIT AUTO FLAGS
3E63: A2 00    01930 INAUTO   LDX #$00
3E65: A9 00    01935          LDA #$00
3E67: 9D 65 3B 01940 .1       STA DL,X
3E6A: 9D 69 3B 01945          STA DR,X
3E6D: 9D 6D 3B 01950          STA DU,X
3E70: 9D 71 3B 01955          STA DD,X
3E73: E8       01960          INX
3E74: E0 04    01965          CPX #$04
3E76: D0 EF    01970          BNE .1
               01975 *START PLAYERS MOVING
3E78: A9 01    01980          LDA #$01
3E7A: 8D 6A 3B 01985          STA DR+1
3E7D: 8D 6B 3B 01990          STA DR+2
3E80: 8D 68 3B 01995          STA DL+3
               02000 *CLEAR P/M AREA
3E83: A9 00    02005 CLEAR    LDA #$00     ;PDATL
3E85: 85 F6    02010          STA PMADR
```

```
3E87: A9 88     02015          LDA /PDATA
3E89: 85 F7     02020          STA PMADR+1
3E8B: A0 00     02025          LDY #$00
3E8D: 98        02030          TYA
3E8E: A2 08     02035          LDX #$08
3E90: 91 F6     02040 .1       STA (PMADR),Y
3E92: C8        02045          INY
3E93: D0 FB     02050          BNE .1
3E95: E6 F7     02055          INC PMADR+1 ;NEXT 256 BYTES
3E97: CA        02060          DEX
3E98: D0 F6     02065          BNE .1
                02070 *PLOT INITIAL PLAYER POSITIONS
3E9A: A2 00     02075          LDX #$00
3E9C: 20 FE 44  02080          JSR PLOTSET0
3E9F: A2 01     02085          LDX #$01
3EA1: 20 03 46  02090          JSR PLOTSET1
                02095 *SET VBLANK
3EA4: A9 07     02100          LDA #$07
3EA6: A2 42     02105          LDX /FRAME  ;HI BYTE VBLANK ROUTINE
3EA8: A0 A1     02110          LDY #FRAME  ;LO BYTE
3EAA: 20 5C E4  02115          JSR SETVBK
                02120 *MAIN CODE LOOP
3EAD: A9 00     02125 LOOPM    LDA #$00
3EAF: 8D 76 3B  02130          STA VBFLAG
                02135 *CHECK COLLISIONS
3EB2: AD 0C D0  02140          LDA POPL    ;CHECK COLLISION PLAYER 0&1
3EB5: C9 02     02145          CMP #$02
3EB7: D0 19     02150          BNE .1
3EB9: 38        02155          SEC
3EBA: AD 62 3B  02160          LDA POINT1  ;IS THERE A 1 LETTER DIFFERENCE
3EBD: ED 61 3B  02165          SBC POINT0
3EC0: C9 02     02170          CMP #$02
3EC2: B0 0E     02175          BGE .1
3EC4: EE 61 3B  02180          INC POINT0  ;PLAYER BECOMES NEXT LETTER
3EC7: EE 62 3B  02185          INC POINT1  ;OLD LETTER JUMPS 2 LETTERS
3ECA: EE 62 3B  02190          INC POINT1
3ECD: A0 01     02195          LDY #$01
3ECF: 20 7A 47  02200          JSR PLACE   ;PUT PLAYER#1 IN NEW PLACE
3ED2: AD 0C D0  02205 .1       LDA POPL    ;CHECK COLLISION PLAYER 0&2
3ED5: C9 04     02210          CMP #$04
3ED7: D0 19     02215          BNE .2
3ED9: 38        02220          SEC
3EDA: AD 63 3B  02225          LDA POINT2  ;IS THERE A 1 LETTER DIFFERENCE
3EDD: ED 61 3B  02230          SBC POINT0
3EE0: C9 02     02235          CMP #$02
3EE2: B0 0E     02240          BGE .2
3EE4: EE 61 3B  02245          INC POINT0
3EE7: EE 63 3B  02250          INC POINT2
3EEA: EE 63 3B  02255          INC POINT2
3EED: A0 02     02260          LDY #$02
3EEF: 20 7A 47  02265          JSR PLACE
3EF2: AD 0C D0  02270 .2       LDA POPL    ;CHECK COLLISION PLAYER 0&3
3EF5: C9 08     02275          CMP #$08
3EF7: D0 58     02280          BNE .3
3EF9: AD 7D 3B  02282          LDA ONCE    ;MUST MOVE PLAYER #3 BEFORE TESTING
3EFC: D0 53     02283          BNE .3                      \FOR NEW COLLISION
3EFE: AD 61 3B  02285          LDA POINT0
3F01: C9 01     02290          CMP #$01
3F03: F0 0C     02295          BEQ .25
3F05: CE 61 3B  02300          DEC POINT0
3F08: CE 62 3B  02305          DEC POINT1
3F0B: CE 63 3B  02310          DEC POINT2
```

```
3F0E: EE 7D 3B 02312          INC ONCE
3F11: A0 03    02315 .25      LDY #$03
3F13: AD 1E 3B 02320          LDA XB       ;PUT PLAYER #3 OPPOSITE SIDE AS PLAYER #0
3F16: C9 0A    02325          CMP #$0A
3F18: B0 1D    02330          BGE .27
3F1A: A9 C8    02335          LDA #$C8     ;PUT PLAYER #3 AT BOTTOM LEFT (200,196)
3F1C: 99 15 3B 02340          STA X,Y
3F1F: A9 C4    02345          LDA #$C4
3F21: 99 1A 3B 02350          STA Y,Y
3F24: A9 00    02355          LDA #$00
3F26: 99 6D 3B 02360          STA DU,Y
3F29: 99 69 3B 02365          STA DR,Y
3F2C: 99 71 3B 02370          STA DD,Y
3F2F: A9 01    02375          LDA #$01
3F31: 99 65 3B 02380          STA DL,Y
3F34: 4C 51 3F 02385          JMP .3
3F37: A9 40    02390 .27      LDA #$40
3F39: 99 15 3B 02395          STA X,Y      ;PUT PLAYER #3 AT BOTTOM RT (48,196)
3F3C: A9 C4    02400          LDA #$C4
3F3E: 99 1A 3B 02405          STA Y,Y
3F41: A9 00    02410          LDA #$00
3F43: 99 6D 3B 02415          STA DU,Y
3F46: 99 71 3B 02420          STA DD,Y
3F49: 99 65 3B 02425          STA DL,Y
3F4C: A9 01    02430          LDA #$01
3F4E: 99 69 3B 02435          STA DR,Y
3F51: 8D 1E D0 02440 .3       STA HITCLR
3F54: A9 00    02445          LDA #$00     ;STOP ATTRACT MODE
3F56: 85 4D    02450          STA $4D
               02455 *PLAYER #1 CODE
3F58: A2 01    02460          LDX #$01
3F5A: 20 06 40 02465          JSR MOVEPL
3F5D: 20 03 46 02470          JSR PLOTSET1
               02475 *PLAYER #2
3F60: A2 02    02480          LDX #$02
3F62: 20 06 40 02485          JSR MOVEPL
3F65: 20 31 46 02490          JSR PLOTSET2
               02495 *PLAYER3
3F68: AD 7A 3B 02500          LDA SLOW
3F6B: C9 00    02505          CMP #$00
3F6D: F0 09    02510          BEQ P3
3F6F: EE 7B 3B 02515          INC HALF     ;INCREMENT COUNTER
3F72: AD 7B 3B 02520          LDA HALF
3F75: 4A       02525          LSR          ;DIVIDE BY 2
3F76: 90 0D    02530          BCC RESET    ;SKIPS ON EVEN CYCLES
3F78: A2 03    02535 P3       LDX #$03
3F7A: 20 06 40 02540          JSR MOVEPL
3F7D: 20 5F 46 02545          JSR PLOTSET3
3F80: A9 00    02546          LDA #$00
3F82: 8D 7D 3B 02547          STA ONCE
               02550 *IF PLAYER #0 BECOMES Z RESET TO SECOND SCREEN
3F85: AD 61 3B 02555 RESET    LDA POINT0
3F88: C9 1A    02560          CMP #$1A
3F8A: D0 6D    02565          BNE FOREVER
3F8C: A9 80    02570          LDA #$80
3F8E: 85 14    02575          STA $14
3F90: A5 14    02580 DELAY    LDA $14      ;ABOUT 2 SEC
3F92: D0 FC    02585          BNE DELAY
               02590 *MOVE PLAYERS TEMP OFF SCREEN
3F94: A9 00    02595          LDA #$00
3F96: 8D C0 02 02600          STA COLPM0   ;TEMP BLACKOUT PLAYER#0 IN VBLANK
3F99: 8D 01 D0 02605          STA HPOSP1
```

```
3F9C: 8D 02 D0 02610          STA HPOSP2
3F9F: 8D 03 D0 02615          STA HPOSP3
                02620 *DONE WITH 2ND LEVEL
3FA2: AD 7C 3B 02625          LDA LEVEL
3FA5: C9 01    02630          CMP #$01
3FA7: D0 03    02635          BNE .1
3FA9: 4C FF 3C 02640          JMP START
                02645 *ERASE SCREEN
3FAC: A2 00    02650 .1       LDX #$00
3FAE: A9 00    02655          LDA #$00
3FB0: 9D 00 87 02660 FLOOP    STA SCREEN,X
3FB3: E8       02665          INX
3FB4: D0 FA    02670          BNE FLOOP
                02675 *WRITE 'NEXT LEVEL'
3FB6: A9 E0    02680          LDA #$E0     ;BACK TO ROM CHARACTER SET
3FB8: 8D F4 02 02685          STA CHBAS
3FBB: EE 7C 3B 02690          INC LEVEL
3FBE: A2 00    02695          LDX #$00
3FC0: BD 00 3B 02700 GLOOP    LDA TITLE2,X
3FC3: 9D 8C 87 02705          STA $878C,X
3FC6: E8       02710          INX
3FC7: E0 14    02715          CPX #$14
3FC9: D0 F5    02720          BNE GLOOP
3FCB: A9 80    02725          LDA #$80
3FCD: 85 14    02730          STA $14      ;2 SECOND DELAY
3FCF: A9 00    02735          LDA #$00
3FD1: 85 13    02740          STA $13
3FD3: A5 13    02745 .1       LDA $13
3FD5: F0 FC    02750          BEQ .1
3FD7: A9 90    02755          LDA /CHRSET ;RAM CHARACTER SET
3FD9: 8D F4 02 02760          STA CHBAS
                02765 *SETUP SECOND SCREEN
3FDC: A2 00    02770          LDX #$00
3FDE: BD 00 39 02775 SLOOP1   LDA DSCREEN2,X
3FE1: 9D 00 87 02780          STA SCREEN,X
3FE4: E8       02785          INX
3FE5: E0 F0    02790          CPX #$F0
3FE7: D0 F5    02795          BNE SLOOP1
3FE9: A9 00    02800          LDA #DSCREEN2
3FEB: 85 F8    02805          STA MAPL
3FED: A9 39    02810          LDA /DSCREEN2
3FEF: 85 F9    02815          STA MAPH
3FF1: A9 7A    02820          LDA #$7A     ;RESTORE PLAYER #0 COLOR
3FF3: 8D C0 02 02822          STA COLPM0
3FF6: 4C C1 3D 02823          JMP RESTART
3FF9: AD 76 3B 02825 FOREVER  LDA VBFLAG  ;STAYS IN THIS LOOP UNTIL
3FFC: C9 01    02830          CMP #$01     ;VBLANK ROUTINE SETS VBFLAG=1
3FFE: D0 03    02835          BNE .1
4000: 4C AD 3E 02840          JMP LOOPM
4003: 4C F9 3F 02845 .1       JMP FOREVER
4006: 20 50 44 02850 MOVEPL   JSR LEGAL
4009: AD 77 3B 02855          LDA LFLAG    ;EQUAL TO 1 IF AT CENTER OF BLOCK
400C: D0 06    02860          BNE .1
400E: 20 B5 45 02865          JSR AUTOP
4011: 4C D1 40 02870          JMP FIN
                02875 *IS THERE AN AVAILABLE DIRECTION OTHER THAN FORWARD OR REVERSE
4014: BD 65 3B 02880 .1       LDA DL,X
4017: F0 2B    02885          BEQ .2
4019: BD 2E 3B 02890          LDA FLAGU,X
401C: D0 03    02895          BNE .11
401E: 4C D4 40 02900          JMP YES
```

```
4021: BD 32 3B 02905 .11      LDA FLAGD,X
4024: DO 03    02910          BNE .12
4026: 4C D4 40 02915          JMP YES
              02920 *CONTINUE MOVING IN SAME DIRECTION UNLESS BLOCKED
4029: BD 26 3B 02925 .12      LDA FLAGL,X
402C: DO 06    02930          BNE .13
402E: DE 15 3B 02935          DEC X,X
4031: 4C D1 40 02940          JMP FIN
4034: A9 00    02945 .13      LDA #$00    ;REVERSE DIRECTION
4036: 9D 65 3B 02950          STA DL,X
4039: A9 01    02955          LDA #$01
403B: 9D 69 3B 02960          STA DR,X
403E: FE 15 3B 02965          INC X,X
4041: 4C D1 40 02970          JMP FIN
4044: BD 69 3B 02975 .2       LDA DR,X
4047: FO 2B    02980          BEQ .3
4049: BD 2E 3B 02985          LDA FLAGU,X
404C: DO 03    02990          BNE .21
404E: 4C D4 40 02995          JMP YES
4051: BD 32 3B 03000 .21      LDA FLAGD,X
4054: DO 03    03005          BNE .22
4056: 4C D4 40 03010          JMP YES
4059: BD 2A 3B 03015 .22      LDA FLAGR,X
405C: DO 06    03020          BNE .23
405E: FE 15 3B 03025          INC X,X
4061: 4C D1 40 03030          JMP FIN
4064: A9 00    03035 .23      LDA #$00    ;REVERSE DIRECTION
4066: 9D 69 3B 03040          STA DR,X
4069: A9 01    03045          LDA #$01
406B: 9D 65 3B 03050          STA DL,X
406E: DE 15 3B 03055          DEC X,X
4071: 4C D1 40 03060          JMP FIN
4074: BD 6D 3B 03065 .3       LDA DU,X
4077: FO 2B    03070          BEQ .4
4079: BD 26 3B 03075          LDA FLAGL,X
407C: DO 03    03080          BNE .31
407E: 4C D4 40 03085          JMP YES
4081: BD 2A 3B 03090 .31      LDA FLAGR,X
4084: DO 03    03095          BNE .32
4086: 4C D4 40 03100          JMP YES
4089: BD 2E 3B 03105 .32      LDA FLAGU,X
408C: DO 06    03110          BNE .33
408E: DE 1A 3B 03115          DEC Y,X
4091: 4C D1 40 03120          JMP FIN
4094: A9 00    03125 .33      LDA #$00    ;REVERSE DIRECTION
4096: 9D 6D 3B 03130          STA DU,X
4099: A9 01    03135          LDA #$01
409B: 9D 71 3B 03140          STA DD,X
409E: FE 1A 3B 03145          INC Y,X
40A1: 4C D1 40 03150          JMP FIN
40A4: BD 71 3B 03155 .4       LDA DD,X
40A7: FO 28    03160          BEQ FIN
40A9: BD 26 3B 03165          LDA FLAGL,X
40AC: DO 03    03170          BNE .41
40AE: 4C D4 40 03175          JMP YES
40B1: BD 2A 3B 03180 .41      LDA FLAGR,X
40B4: DO 03    03185          BNE .42
40B6: 4C D4 40 03190          JMP YES
40B9: BD 32 3B 03195 .42      LDA FLAGD,X
40BC: DO 06    03200          BNE .43
40BE: FE 1A 3B 03205          INC Y,X
```

```
40C1: 4C D1 40 03210          JMP FIN
40C4: A9 00    03215 .43      LDA #$00    ;REVERSE DIRECTION
40C6: 9D 71 3B 03220          STA DD,X
40C9: A9 01    03225          LDA #$01
40CB: 9D 6D 3B 03230          STA DU,X
40CE: DE 1A 3B 03235          DEC Y,X
40D1: 4C A0 42 03240 FIN      JMP EE
40D4: EA       03245 YES      NOP
               03250 *SET ALL REL,X FLAGS
               03255 *VALUE OF 0 WOULDN'T WANT TO MOVE BUT VALUE OF 1 WOULD HOME
               03260 *PLAYER 1&2 FLEES - PLAYER 3 CHASES          \IN ON PLAYER
40D5: A9 00    03265 TESTRX   LDA #$00    ;ZERO REL,X FLAGS
40D7: 9D 36 3B 03270          STA RELL,X
40DA: 9D 3A 3B 03275          STA RELR,X
40DD: 9D 42 3B 03280          STA RELD,X
40E0: 9D 3E 3B 03285          STA RELU,X
40E3: BD 1E 3B 03290          LDA XB,X
40E6: 38       03295          SEC
40E7: ED 1E 3B 03300          SBC XB
40EA: F0 2A    03302          BEQ .4
40EC: 10 14    03305          BPL .1
40EE: E0 03    03310          CPX #$03
40F0: F0 08    03315          BEQ .2
40F2: A9 01    03320          LDA #$01
40F4: 9D 36 3B 03325          STA RELL,X  ;ACTUALLY RELR,X=-1 BUT RELL,X=1
40F7: 4C 22 41 03330          JMP TESTRY
40FA: A9 01    03335 .2       LDA #$01
40FC: 9D 3A 3B 03340          STA RELR,X
40FF: 4C 22 41 03345          JMP TESTRY
4102: E0 03    03350 .1       CPX #$03
4104: F0 08    03355          BEQ .3
4106: A9 01    03360          LDA #$01
4108: 9D 3A 3B 03365          STA RELR,X  ;ACTUALLY RELL,X=-1
410B: 4C 22 41 03370          JMP TESTRY
410E: A9 01    03375 .3       LDA #$01
4110: 9D 36 3B 03376          STA RELL,X
4113: 4C 22 41 03377          JMP TESTRY
4116: E0 03    03378 .4       CPX #$03
4118: F0 08    03379          BEQ TESTRY
411A: A9 01    03380          LDA #$01
411C: 9D 36 3B 03381          STA RELL,X
411F: 9D 3A 3B 03382          STA RELR,X
4122: BD 22 3B 03385 TESTRY   LDA YB,X
4125: 38       03390          SEC
4126: ED 22 3B 03395          SBC YB
4129: 10 16    03400          BPL .1
412B: F0 2B    03402          BEQ .4
412D: E0 03    03405          CPX #$03
412F: F0 08    03410          BEQ .2
4131: A9 01    03415          LDA #$01
4133: 9D 3E 3B 03420          STA RELU,X  ;ACTUALLY RELD,X=-1
4136: 4C 64 41 03425          JMP MATCHT
4139: A9 01    03430 .2       LDA #$01
413B: 9D 42 3B 03435          STA RELD,X
413E: 4C 64 41 03440          JMP MATCHT
4141: E0 03    03445 .1       CPX #$03
4143: F0 08    03450          BEQ .3
4145: A9 01    03455          LDA #$01
4147: 9D 42 3B 03460          STA RELD,X  ;ACTUALLY RELU,X=-1
414A: 4C 64 41 03465          JMP MATCHT
414D: A9 01    03470 .3       LDA #$01
```

```
414F: 9D 3E 3B 03475          STA RELU,X
4152: 9D 3E 3B 03476          STA RELU,X
4155: 4C 64 41 03477          JMP MATCHT
4158: E0 03    03478 .4       CPX #$03
415A: F0 08    03479          BEQ MATCHT
415C: A9 01    03480          LDA #$01
415E: 9D 42 3B 03481          STA RELD,X
4161: 9D 3E 3B 03482          STA RELU,X
4164: A9 00    03485 MATCHT   LDA #$00
4166: 8D 78 3B 03490          STA NUM
4169: 9D 46 3B 03495          STA MFLAGL,X
416C: 9D 4A 3B 03500          STA MFLAGR,X
416F: 9D 4E 3B 03505          STA MFLAGU,X
4172: 9D 52 3B 03510          STA MFLAGD,X
4175: BD 26 3B 03515          LDA FLAGL,X
4178: D0 0F    03520          BNE .2
417A: BD 36 3B 03525          LDA RELL,X
417D: C9 01    03530          CMP #$01
417F: D0 08    03535          BNE .2
4181: EE 78 3B 03540          INC NUM
4184: A9 01    03545          LDA #$01
4186: 9D 46 3B 03550          STA MFLAGL,X ;SET MATCH FLAG
4189: BD 2A 3B 03555 .2       LDA FLAGR,X
418C: D0 0F    03560          BNE .3
418E: BD 3A 3B 03565          LDA RELR,X
4191: C9 01    03570          CMP #$01
4193: D0 08    03575          BNE .3
4195: EE 78 3B 03580          INC NUM
4198: A9 01    03585          LDA #$01
419A: 9D 4A 3B 03590          STA MFLAGR,X
419D: BD 2E 3B 03595 .3       LDA FLAGU,X
41A0: D0 0F    03600          BNE .4
41A2: BD 3E 3B 03605          LDA RELU,X
41A5: C9 01    03610          CMP #$01
41A7: D0 08    03615          BNE .4
41A9: EE 78 3B 03620          INC NUM
41AC: A9 01    03625          LDA #$01
41AE: 9D 4E 3B 03630          STA MFLAGU,X
41B1: BD 32 3B 03635 .4       LDA FLAGD,X
41B4: D0 0F    03640          BNE PP
41B6: BD 42 3B 03645          LDA RELD,X
41B9: C9 01    03650          CMP #$01
41BB: D0 08    03655          BNE PP
41BD: EE 78 3B 03660          INC NUM
41C0: A9 01    03665          LDA #$01
41C2: 9D 52 3B 03670          STA MFLAGD,X
               03675 *IF MATCH CAUSES PLAYER TO REVERSE PREVENT IT
41C5: BD 65 3B 03680 PP       LDA DL,X
41C8: F0 10    03685          BEQ .42
41CA: BD 4A 3B 03690          LDA MFLAGR,X
41CD: F0 0B    03695          BEQ .42
41CF: A9 00    03700          LDA #$00
41D1: 9D 4A 3B 03705          STA MFLAGR,X
41D4: CE 78 3B 03710          DEC NUM
41D7: 4C 16 42 03715          JMP .5
41DA: BD 69 3B 03720 .42      LDA DR,X
41DD: F0 10    03725          BEQ .43
41DF: BD 46 3B 03730          LDA MFLAGL,X
41E2: F0 0B    03735          BEQ .43
41E4: A9 00    03740          LDA #$00
41E6: 9D 46 3B 03745          STA MFLAGL,X
```

```
41E9: CE 78 3B 03750          DEC NUM
41EC: 4C 16 42 03755          JMP .5
41EF: BD 6D 3B 03760 .43      LDA DU,X
41F2: F0 10    03765          BEQ .44
41F4: BD 52 3B 03770          LDA MFLAGD,X
41F7: F0 0B    03775          BEQ .44
41F9: A9 00    03780          LDA #$00
41FB: 9D 52 3B 03785          STA MFLAGD,X
41FE: CE 78 3B 03790          DEC NUM
4201: 4C 16 42 03795          JMP .5
4204: BD 71 3B 03800 .44      LDA DD,X
4207: F0 0D    03805          BEQ .5
4209: BD 4E 3B 03810          LDA MFLAGU,X
420C: F0 08    03815          BEQ .5
420E: A9 00    03820          LDA #$00
4210: 9D 4E 3B 03825          STA MFLAGU,X
4213: CE 78 3B 03830          DEC NUM
               03835 *IF TWO MATCHES THEN CHOOSE A DIRECTION RANDOMILY -
4216: AD 78 3B 03840 .5       LDA NUM              THEN MOVE IN DIRECTION
4219: D0 06    03845          BNE .55
421B: 20 8D 46 03850          JSR CORECT
421E: 4C A0 42 03855          JMP EE
4221: AD 78 3B 03860 .55      LDA NUM
4224: C9 01    03865          CMP #$01
4226: F0 07    03870          BEQ .6
4228: AD 0A D2 03875          LDA RANDOM
422B: C9 50    03880          CMP #80
422D: 90 3A    03885          BLT .8
422F: BD 46 3B 03890 .6       LDA MFLAGL,X
4232: C9 01    03895          CMP #$01
4234: D0 16    03900          BNE .7
4236: A9 01    03905          LDA #$01
4238: 9D 65 3B 03910          STA DL,X
423B: A9 00    03915          LDA #$00
423D: 9D 71 3B 03920          STA DD,X
4240: 9D 69 3B 03925          STA DR,X
4243: 9D 6D 3B 03930          STA DU,X
4246: DE 15 3B 03935          DEC X,X
4249: 4C A0 42 03940          JMP EE
424C: BD 4A 3B 03945 .7       LDA MFLAGR,X
424F: C9 01    03950          CMP #$01
4251: D0 16    03955          BNE .8
4253: A9 01    03960          LDA #$01
4255: 9D 69 3B 03965          STA DR,X
4258: A9 00    03970          LDA #$00
425A: 9D 65 3B 03975          STA DL,X
425D: 9D 6D 3B 03980          STA DU,X
4260: 9D 71 3B 03985          STA DD,X
4263: FE 15 3B 03990          INC X,X
4266: 4C A0 42 03995          JMP EE
4269: BD 4E 3B 04000 .8       LDA MFLAGU,X
426C: C9 01    04005          CMP #$01
426E: D0 16    04010          BNE .9
4270: A9 01    04015          LDA #$01
4272: 9D 6D 3B 04020          STA DU,X
4275: A9 00    04025          LDA #$00
4277: 9D 65 3B 04030          STA DL,X
427A: 9D 69 3B 04035          STA DR,X
427D: 9D 71 3B 04040          STA DD,X
4280: DE 1A 3B 04045          DEC Y,X
4283: 4C A0 42 04050          JMP EE
4286: BD 52 3B 04055 .9       LDA MFLAGD,X
```

```
4289: C9 01     04060          CMP #$01
428B: D0 13     04065          BNE EE
428D: A9 01     04070          LDA #$01
428F: 9D 71 3B  04075          STA DD,X
4292: A9 00     04080          LDA #$00
4294: 9D 6D 3B  04085          STA DU,X
4297: 9D 65 3B  04090          STA DL,X
429A: 9D 69 3B  04095          STA DR,X
429D: FE 1A 3B  04100          INC Y,X
42A0: 60        04105 EE       RTS
                04110 *VBLANK ROUTINE
42A1: EA        04115 FRAME    NOP
42A2: AD FC 02  04120 PAUSE1   LDA $2FC    ;KEY PRESSED?
42A5: C9 FF     04125          CMP #$FF
42A7: F0 25     04130          BEQ .2
42A9: C9 21     04135          CMP #$21    ;SPACE BAR?
42AB: D0 2B     04140          BNE .3
42AD: AD 79 3B  04145          LDA PAUSE   ;PAUSE=1?
42B0: C9 01     04150          CMP #$01
42B2: F0 0D     04155          BEQ .1
42B4: A9 01     04160          LDA #$01    ;NOT PAUSED-THEN SET PAUSE
42B6: 8D 79 3B  04165          STA PAUSE
42B9: A9 FF     04170          LDA #$FF
42BB: 8D FC 02  04175          STA $2FC
42BE: 4C 62 E4  04180          JMP XITVBK
42C1: A9 00     04185 .1       LDA #$00    ;PAUSED-THEN RELEASE PAUSE
42C3: 8D 79 3B  04190          STA PAUSE
42C6: A9 FF     04195          LDA #$FF
42C8: 8D FC 02  04200          STA $2FC
42CB: 4C D8 42  04205          JMP .3
42CE: AD 79 3B  04210 .2       LDA PAUSE   ;PAUSED?
42D1: C9 01     04215          CMP #$01
42D3: D0 03     04220          BNE .3
42D5: 4C 62 E4  04225          JMP XITVBK
42D8: A9 01     04230 .3       LDA #01
42DA: 8D 76 3B  04235          STA VBFLAG
42DD: A2 00     04240          LDX #$00
42DF: 20 50 44  04245          JSR LEGAL
                04250 *READ JOYSTICK
42E2: AD 78 02  04255          LDA STICK   ;JOYSTICK CENTERED?
42E5: C9 0F     04260          CMP #$0F
42E7: D0 06     04265          BNE DUR
42E9: 20 32 45  04270          JSR AUTO    ;IF STICK NOT TOUCHED CONTINUE IN SAME DIR
42EC: 4C 43 44  04275          JMP STAY
42EF: AD 78 02  04280 DUR      LDA STICK   ;TEST DIAGONAL UP&RT
42F2: C9 06     04285          CMP #$06
42F4: D0 1B     04290          BNE DDR
42F6: AD 69 3B  04295          LDA DR
42F9: F0 03     04300          BEQ .01
42FB: 4C E1 43  04305          JMP CHKUP
42FE: AD 6D 3B  04310 .01      LDA DU
4301: D0 06     04315          BNE .02
4303: 20 32 45  04320          JSR AUTO
4306: 4C 43 44  04325          JMP STAY
4309: A9 01     04330 .02      LDA #$01    ;SET TO RT ONLY
430B: 8D 75 3B  04335          STA INHIBIT
430E: 4C 77 43  04340          JMP CHKRT
4311: AD 78 02  04345 DDR      LDA STICK   ;TEST DIAGONAL DOWN&RT
4314: C9 05     04350          CMP #$05
4316: D0 1B     04355          BNE DUL
4318: AD 69 3B  04360          LDA DR
431B: F0 03     04365          BEQ .03
```

```
431D: 4C 12 44 04370          JMP CHKDN
4320: AD 71 3B 04375 .03      LDA DD
4323: D0 06    04380          BNE .04
4325: 20 32 45 04385          JSR AUTO
4328: 4C 43 44 04390          JMP STAY
432B: A9 01    04395 .04      LDA #$01     ;SET TO RT ONLY
432D: 8D 75 3B 04400          STA INHIBIT
4330: 4C 77 43 04405          JMP CHKRT
4333: AD 78 02 04410 DUL      LDA STICK    ;TEST DIAGONAL UP&LEFT
4336: C9 0A    04415          CMP #$0A
4338: D0 1B    04420          BNE DDL
433A: AD 65 3B 04425          LDA DL
433D: F0 03    04430          BEQ .05
433F: 4C E1 43 04435          JMP CHKUP
4342: AD 6D 3B 04440 .05      LDA DU
4345: D0 06    04445          BNE .06
4347: 20 32 45 04450          JSR AUTO
434A: 4C 43 44 04455          JMP STAY
434D: A9 01    04460 .06      LDA #$01     ;SET TO LEFT ONLY
434F: 8D 75 3B 04465          STA INHIBIT
4352: 4C A8 43 04470          JMP CHKLF
4355: AD 78 02 04475 DDL      LDA STICK    ;TEST DIAGONAL DOWN&LEFT
4358: C9 09    04480          CMP #$09
435A: D0 1B    04485          BNE CHKRT
435C: AD 65 3B 04490          LDA DL
435F: F0 03    04495          BEQ .07
4361: 4C 12 44 04500          JMP CHKDN
4364: AD 71 3B 04505 .07      LDA DD
4367: D0 06    04510          BNE .08
4369: 20 32 45 04515          JSR AUTO
436C: 4C 43 44 04520          JMP STAY
436F: A9 01    04525 .08      LDA #$01     ;SET TO LEFT ONLY
4371: 8D 75 3B 04530          STA INHIBIT
4374: 4C A8 43 04535          JMP CHKLF
4377: AD 78 02 04540 CHKRT    LDA STICK
437A: 29 08    04545          AND #$08     ;RT BIT?
437C: D0 2A    04550          BNE CHKLF
437E: AD 2A 3B 04555          LDA FLAGR
4381: F0 06    04560          BEQ .2
4383: 20 32 45 04565          JSR AUTO
4386: 4C 43 44 04570          JMP STAY
4389: A9 00    04575 .2       LDA #$00
438B: 8D 65 3B 04580          STA DL       ;SHUT OFF AUTO FLAGS
438E: 8D 6D 3B 04585          STA DU
4391: 8D 71 3B 04590          STA DD
4394: 8D 26 3B 04595          STA FLAGL    ;RESET LEGAL FLAGS-ONLY CAN TRAVEL
4397: A9 01    04600          LDA #$01                    \BETWEEN TWO BLOCKS
4399: 8D 2E 3B 04605          STA FLAGU
439C: 8D 32 3B 04610          STA FLAGD
439F: 8D 69 3B 04615          STA DR       ;TURN ON AUTOMATIC FLAG
43A2: EE 14 3B 04620          INC XO       ;XO=XO+1
43A5: EE 15 3B 04625          INC X
43A8: AD 78 02 04630 CHKLF    LDA STICK
43AB: 29 04    04635          AND #$04     ;LEFT BIT
43AD: D0 2A    04640          BNE CHK
43AF: AD 26 3B 04645          LDA FLAGL
43B2: F0 06    04650          BEQ .3
43B4: 20 32 45 04655          JSR AUTO
43B7: 4C 43 44 04660          JMP STAY
43BA: A9 00    04665 .3       LDA #$00
43BC: 8D 69 3B 04670          STA DR
43BF: 8D 6D 3B 04675          STA DU
```

```
43C2: 8D 71 3B 04680          STA DD
43C5: 8D 2A 3B 04685          STA FLAGR
43C8: A9 01    04690          LDA #$01
43CA: 8D 2E 3B 04695          STA FLAGU
43CD: 8D 32 3B 04700          STA FLAGD
43D0: 8D 65 3B 04705          STA DL
43D3: CE 14 3B 04710          DEC XO       ;XO=XO-1
43D6: CE 15 3B 04715          DEC X
43D9: AD 75 3B 04720 CHK      LDA INHIBIT ;THIS PREVENTS GETTING BOTH DIRECTIONS
43DC: F0 03    04725          BEQ CHKUP                                  ON DIAGONAL
43DE: 4C 43 44 04730          JMP STAY
43E1: AD 78 02 04735 CHKUP    LDA STICK
43E4: 29 01    04740          AND #$01     ;UP BIT?
43E6: D0 2A    04745          BNE CHKDN
43E8: AD 2E 3B 04750          LDA FLAGU
43EB: F0 06    04755          BEQ .4
43ED: 20 32 45 04760          JSR AUTO
43F0: 4C 43 44 04765          JMP STAY
43F3: A9 00    04770 .4       LDA #$00
43F5: 8D 71 3B 04775          STA DD
43F8: 8D 69 3B 04780          STA DR
43FB: 8D 65 3B 04785          STA DL
43FE: 8D 32 3B 04790          STA FLAGD
4401: A9 01    04795          LDA #$01
4403: 8D 26 3B 04800          STA FLAGL
4406: 8D 2A 3B 04805          STA FLAGR
4409: 8D 6D 3B 04810          STA DU
440C: CE 19 3B 04815          DEC YO       ;YO=YO-1
440F: CE 1A 3B 04820          DEC Y
4412: AD 78 02 04825 CHKDN    LDA STICK
4415: 29 02    04830          AND #$02
4417: D0 2A    04835          BNE STAY
4419: AD 32 3B 04840          LDA FLAGD
441C: F0 06    04845          BEQ .5
441E: 20 32 45 04850          JSR AUTO
4421: 4C 43 44 04855          JMP STAY
4424: A9 00    04860 .5       LDA #$00
4426: 8D 6D 3B 04865          STA DU
4429: 8D 69 3B 04870          STA DR
442C: 8D 65 3B 04875          STA DL
442F: 8D 2E 3B 04880          STA FLAGU
4432: A9 01    04885          LDA #$01
4434: 8D 26 3B 04890          STA FLAGL
4437: 8D 2A 3B 04895          STA FLAGR
443A: 8D 71 3B 04900          STA DD
443D: EE 19 3B 04905          INC YO       ;YO=YO+1
4440: EE 1A 3B 04910          INC Y
4443: A9 00    04915 STAY     LDA #$00
4445: 8D 75 3B 04920          STA INHIBIT
4448: A2 00    04925          LDX #$00
444A: 20 FE 44 04930          JSR PLOTSETO
444D: 4C 62 E4 04935          JMP XITVBK
               04940 *SUBROUTINE-TEST FOR LEGAL MOVES
               04945 *INPUT X,Y PLAYER POSITION; X REG - PLAYER #
               04950 *OUTPUT FLAGL,X;FLAGR,X;FLAGU,X;FLAGD,X--0-OPEN,1-CLOSE
4450: A9 00    04955 LEGAL    LDA #$00
4452: 8D 77 3B 04960          STA LFLAG
4455: BD 15 3B 04965          LDA X,X      ;PLAYER HOR POSITION
4458: 38       04970          SEC
4459: E9 30    04975          SBC #$30     ;SUBTRACT 48
445B: 8D 56 3B 04980          STA TEMPX
445E: 4A       04985          LSR          ;DIVIDE BY 8
```

```
445F: 4A        04990          LSR
4460: 4A        04995          LSR
4461: 9D 1E 3B  05000          STA XB,X
4464: BD 1A 3B  05005          LDA Y,X      ;PLAYER VERT POSITION
4467: 38        05010          SEC
4468: E9 24     05015          SBC #$24     ;SUBTRACT 36
446A: 8D 57 3B  05020          STA TEMPY
446D: 4A        05025          LSR          ;DIVIDE BY 16
446E: 4A        05030          LSR
446F: 4A        05035          LSR
4470: 4A        05040          LSR
4471: 9D 22 3B  05045          STA YB,X
                05050 *TEST IF AT CENTER OF BLOCK
4474: AD 57 3B  05055          LDA TEMPY
4477: 29 0F     05060          AND #$0F     ;TEST AGAINST FIRST 4 BITS-ONLY MULTIPLE
4479: D0 61     05065          BNE DONE                                  \OF 16 @CENTER
447B: AD 56 3B  05070          LDA TEMPX
447E: 29 07     05075          AND #$07     ;TEST AGAINST FIRST 3 BITS-ONLY MULTIPLE
4480: D0 5A     05080          BNE DONE                                  \OF 8 @CENTER
4482: A9 01     05085          LDA #$01
4484: 8D 77 3B  05090          STA LFLAG
                05095 *AT CENTER-TEST&RESET LEGAL MOVE FLAGS
                05100 *BLOCK=(YB*20)+XB
4487: A9 00     05105          LDA #$00
4489: BC 22 3B  05110          LDY YB,X
448C: C0 00     05115          CPY #$00     ;IF 0 SKIP ADD & DECREMENT
448E: F0 08     05120          BEQ .5
4490: 18        05125 .1       CLC
4491: 69 14     05130          ADC #$14
4493: 88        05135          DEY
4494: C0 00     05140          CPY #$00
4496: D0 F8     05145          BNE .1
4498: 18        05150 .5       CLC
4499: 7D 1E 3B  05155          ADC XB,X
449C: 8D 60 3B  05160          STA BLOCK
449F: A9 00     05165          LDA #$00     ;OPEN ALL FLAGS FOR PLAYER
44A1: 9D 26 3B  05170          STA FLAGL,X
44A4: 9D 2A 3B  05175          STA FLAGR,X
44A7: 9D 2E 3B  05180          STA FLAGU,X
44AA: 9D 32 3B  05185          STA FLAGD,X
                05190 *TEST BLOCK MAN IN FOR LEGAL MOVES
44AD: AC 60 3B  05195          LDY BLOCK
44B0: B1 F8     05200          LDA (MAPL),Y
44B2: 29 01     05205          AND #$01
44B4: F0 05     05210          BEQ .2
44B6: A9 01     05215          LDA #$01
44B8: 9D 26 3B  05220          STA FLAGL,X
44BB: B1 F8     05225 .2       LDA (MAPL),Y
44BD: 29 02     05230          AND #$02
44BF: F0 05     05235          BEQ .3
44C1: A9 01     05240          LDA #$01
44C3: 9D 2A 3B  05245          STA FLAGR,X
44C6: B1 F8     05250 .3       LDA (MAPL),Y
44C8: 29 04     05255          AND #$04
44CA: F0 05     05260          BEQ .4
44CC: A9 01     05265          LDA #$01
44CE: 9D 2E 3B  05270          STA FLAGU,X
44D1: B1 F8     05275 .4       LDA (MAPL),Y
44D3: 29 08     05280          AND #$08
44D5: F0 05     05285          BEQ DONE
44D7: A9 01     05290          LDA #$01
44D9: 9D 32 3B  05295          STA FLAGD,X
```

```
44DC: 60        05300 DONE     RTS
                05305 *PUT SHAPE IN P/M AREA
44DD: A0 00     05310 PLOT     LDY #$00     ;COUNTER
44DF: A9 00     05315          LDA #$00     ;NEED 0 TO ERASE EACH TIME
44E1: 91 F4     05320 .1       STA (SHPMOL),Y;ERASE OLD SHAPE FIRST
44E3: C8        05325          INY
44E4: C0 08     05330          CPY #$08
44E6: 90 F9     05335          BLT .1
44E8: A0 00     05340          LDY #$00
44EA: B1 F0     05345 .2       LDA (SHPL),Y;GET BYTE FROM PROPER SHAPE TABLE
44EC: 91 F2     05350          STA (SHPML),Y ;PUT IN P/M AREA
44EE: C8        05355          INY
44EF: C0 08     05360          CPY #$08
44F1: 90 F7     05365          BLT .2
44F3: A5 F2     05370          LDA SHPML    ;TRANSFER NEW P/M POS TO OLD POS
44F5: 9D 58 3B  05375          STA TEMPL,X
44F8: A5 F3     05380          LDA SHPMH
44FA: 9D 5C 3B  05385          STA TEMPH,X
44FD: 60        05390          RTS
                05395 *SUBROUTINE PLOTSETO
44FE: AD 14 3B  05400 PLOTSETO LDA XO
4501: 8D 15 3B  05405          STA X
4504: 8D 00 D0  05410          STA HPOSP0
4507: AD 19 3B  05415          LDA YO
450A: 8D 1A 3B  05420          STA Y
450D: 85 F2     05425          STA SHPML
450F: A9 88     05430          LDA /PDATA
4511: 18        05435          CLC
4512: 69 04     05440          ADC #$04     ;PLAYER IS 1K BEYOND START
4514: 85 F3     05445          STA SHPMH
4516: AC 61 3B  05450          LDY POINTO
4519: B9 7E 3B  05455          LDA SHPLO,Y ;POINTER TO CORRECT SHAPE
451C: 85 F0     05460          STA SHPL
451E: A9 3C     05465          LDA /SHAPES
4520: 85 F1     05470          STA SHPH
4522: A2 00     05475          LDX #$00
4524: BD 58 3B  05480          LDA TEMPL,X
4527: 85 F4     05485          STA SHPMOL
4529: BD 5C 3B  05490          LDA TEMPH,X
452C: 85 F5     05495          STA SHPMOH
452E: 20 DD 44  05500          JSR PLOT
4531: 60        05505          RTS
                05510 *CONTINUE MOVING IN LAST DIR UNLESS HIT WALL
                05515 *AUTO ON - #1
4532: AD 69 3B  05520 AUTO     LDA DR
4535: C9 01     05525          CMP #$01
4537: D0 1A     05530          BNE .1
4539: AD 2A 3B  05535          LDA FLAGR
453C: C9 00     05540          CMP #$00
453E: D0 73     05545          BNE .4
4540: A9 00     05550          LDA #$00
4542: 8D 26 3B  05555          STA FLAGL
4545: A9 01     05560          LDA #$01
4547: 8D 2E 3B  05565          STA FLAGU
454A: 8D 32 3B  05570          STA FLAGD
454D: EE 14 3B  05575          INC XO
4550: 4C B3 45  05580          JMP .4
4553: AD 65 3B  05585 .1       LDA DL
4556: C9 01     05590          CMP #$01
4558: D0 1A     05595          BNE .2
455A: AD 26 3B  05600          LDA FLAGL
455D: C9 00     05605          CMP #$00
```

```
455F: D0 52    05610          BNE .4
4561: A9 00    05615          LDA #$00
4563: 8D 2A 3B 05620          STA FLAGR
4566: A9 01    05625          LDA #$01
4568: 8D 2E 3B 05630          STA FLAGU
456B: 8D 32 3B 05635          STA FLAGD
456E: CE 14 3B 05640          DEC XO
4571: 4C B3 45 05645          JMP .4
4574: AD 6D 3B 05650 .2       LDA DU
4577: C9 01    05655          CMP #$01
4579: D0 1A    05660          BNE .3
457B: AD 2E 3B 05665          LDA FLAGU
457E: C9 00    05670          CMP #$00
4580: D0 31    05675          BNE .4
4582: A9 00    05680          LDA #$00
4584: 8D 32 3B 05685          STA FLAGD
4587: A9 01    05690          LDA #$01
4589: 8D 2A 3B 05695          STA FLAGR
458C: 8D 26 3B 05700          STA FLAGL
458F: CE 19 3B 05705          DEC YO
4592: 4C B3 45 05710          JMP .4
4595: AD 71 3B 05715 .3       LDA DD
4598: C9 01    05720          CMP #$01
459A: D0 17    05725          BNE .4
459C: AD 32 3B 05730          LDA FLAGD
459F: C9 00    05735          CMP #$00
45A1: D0 10    05740          BNE .4
45A3: A9 00    05745          LDA #$00
45A5: 8D 2E 3B 05750          STA FLAGU
45A8: A9 01    05755          LDA #$01
45AA: 8D 2A 3B 05760          STA FLAGR
45AD: 8D 26 3B 05765          STA FLAGL
45B0: EE 19 3B 05770          INC YO
45B3: EA       05775 .4       NOP
45B4: 60       05780          RTS
               05785 *AUTO CODE FOR PLAYERS 1,2 & 3
               05790 *NEEDS PLAYER # IN X-REGISTER
               05795 *CODE DOESN'T REQUIRE LATCHES SINCE JOYSTICK CAN'T SEND
45B5: BD 69 3B 05800 AUTOP    LDA DR,X                \THOSE PLAYERS ORDERS
45B8: C9 01    05805          CMP #$01
45BA: D0 0D    05810          BNE .1
45BC: BD 2A 3B 05815          LDA FLAGR,X
45BF: C9 00    05820          CMP #$00
45C1: D0 3F    05825          BNE .4
45C3: FE 15 3B 05830          INC X,X
45C6: 4C 02 46 05835          JMP .4
45C9: BD 65 3B 05840 .1       LDA DL,X
45CC: C9 01    05845          CMP #$01
45CE: D0 0D    05850          BNE .2
45D0: BD 26 3B 05855          LDA FLAGL,X
45D3: C9 00    05860          CMP #$00
45D5: D0 2B    05865          BNE .4
45D7: DE 15 3B 05870          DEC X,X
45DA: 4C 02 46 05875          JMP .4
45DD: BD 6D 3B 05880 .2       LDA DU,X
45E0: C9 01    05885          CMP #$01
45E2: D0 0D    05890          BNE .3
45E4: BD 2E 3B 05895          LDA FLAGU,X
45E7: C9 00    05900          CMP #$00
45E9: D0 17    05905          BNE .4
45EB: DE 1A 3B 05910          DEC Y,X
45EE: 4C 02 46 05915          JMP .4
```

```
45F1: BD 71 3B 05920 .3       LDA DD,X
45F4: C9 01    05925          CMP #$01
45F6: D0 0A    05930          BNE .4
45F8: BD 32 3B 05935          LDA FLAGD,X
45FB: C9 00    05940          CMP #$00
45FD: D0 03    05945          BNE .4
45FF: FE 1A 3B 05950          INC Y,X
4602: 60       05955 .4       RTS
               05960 *SUBROUTINE PLOTSET1
4603: BD 15 3B 05965 PLOTSET1 LDA X,X
4606: 8D 01 D0 05970          STA HPOSP1
4609: BD 1A 3B 05975          LDA Y,X
460C: 85 F2    05980          STA SHPML
460E: A9 88    05985          LDA /PDATA
4610: 18       05990          CLC
4611: 69 05    05995          ADC #$05    ;PLAYER IS 1.25K BEYOND START
4613: 85 F3    06000          STA SHPMH
4615: AC 62 3B 06005          LDY POINT1
4618: B9 7E 3B 06010          LDA SHPLO,Y ;POINTER TO CORRECT SHAPE
461B: 85 F0    06015          STA SHPL
461D: A9 3C    06020          LDA /SHAPES
461F: 85 F1    06025          STA SHPH
4621: A2 01    06030          LDX #$01
4623: BD 58 3B 06035          LDA TEMPL,X
4626: 85 F4    06040          STA SHPMOL
4628: BD 5C 3B 06045          LDA TEMPH,X
462B: 85 F5    06050          STA SHPMOH
462D: 20 DD 44 06055          JSR PLOT
4630: 60       06060          RTS
               06065 *SUBROUTINE PLOTSET2
4631: BD 15 3B 06070 PLOTSET2 LDA X,X
4634: 8D 02 D0 06075          STA HPOSP2
4637: BD 1A 3B 06080          LDA Y,X
463A: 85 F2    06085          STA SHPML
463C: A9 88    06090          LDA /PDATA
463E: 18       06095          CLC
463F: 69 06    06100          ADC #$06    ;PLAYER IS 1.5K BEYOND START
4641: 85 F3    06105          STA SHPMH
4643: AC 63 3B 06110          LDY POINT2
4646: B9 7E 3B 06115          LDA SHPLO,Y ;POINTER TO CORRECT SHAPE
4649: 85 F0    06120          STA SHPL
464B: A9 3C    06125          LDA /SHAPES
464D: 85 F1    06130          STA SHPH
464F: A2 02    06135          LDX #$02
4651: BD 58 3B 06140          LDA TEMPL,X
4654: 85 F4    06145          STA SHPMOL
4656: BD 5C 3B 06150          LDA TEMPH,X
4659: 85 F5    06155          STA SHPMOH
465B: 20 DD 44 06160          JSR PLOT
465E: 60       06165          RTS
               06170 *SUBROUTINE PLOTSET3
465F: BD 15 3B 06175 PLOTSET3 LDA X,X
4662: 8D 03 D0 06180          STA HPOSP3
4665: BD 1A 3B 06185          LDA Y,X
4668: 85 F2    06190          STA SHPML
466A: A9 88    06195          LDA /PDATA
466C: 18       06200          CLC
466D: 69 07    06205          ADC #$07    ;PLAYER IS 1.75K BEYOND START
466F: 85 F3    06210          STA SHPMH
4671: AC 64 3B 06215          LDY POINT3
4674: B9 7E 3B 06220          LDA SHPLO,Y ;POINTER TO CORRECT SHAPE
4677: 85 F0    06225          STA SHPL
```

```
4679: A9 3C    06230          LDA /SHAPES
467B: 85 F1    06235          STA SHPH
467D: A2 03    06240          LDX #$03
467F: BD 58 3B 06245          LDA TEMPL,X
4682: 85 F4    06250          STA SHPMOL
4684: BD 5C 3B 06255          LDA TEMPH,X
4687: 85 F5    06260          STA SHPMOH
4689: 20 DD 44 06265          JSR PLOT
468C: 60       06270          RTS
468D: BD 69 3B 06275 CORECT   LDA DR,X
4690: F0 35    06280          BEQ .2
4692: BD 2A 3B 06285 .11      LDA FLAGR,X
4695: D0 06    06290          BNE .12
4697: 20 B5 45 06295          JSR AUTOP
469A: 4C 79 47 06300          JMP FF
469D: BD 2E 3B 06305 .12      LDA FLAGU,X
46A0: D0 10    06310          BNE .13
46A2: DE 1A 3B 06315          DEC Y,X
46A5: A9 00    06320          LDA #$00
46A7: 9D 69 3B 06325          STA DR,X
46AA: A9 01    06330          LDA #$01
46AC: 9D 6D 3B 06335          STA DU,X
46AF: 4C 79 47 06340          JMP FF
46B2: BD 32 3B 06345 .13      LDA FLAGD,X
46B5: D0 0D    06350          BNE .14
46B7: FE 1A 3B 06355          INC Y,X
46BA: A9 00    06360          LDA #$00
46BC: 9D 69 3B 06365          STA DR,X
46BF: A9 01    06370          LDA #$01
46C1: 9D 71 3B 06375          STA DD,X
46C4: 4C 79 47 06380 .14      JMP FF
46C7: BD 65 3B 06385 .2       LDA DL,X
46CA: F0 35    06390          BEQ .3
46CC: BD 26 3B 06395 .21      LDA FLAGL,X
46CF: D0 06    06400          BNE .22
46D1: 20 B5 45 06405          JSR AUTOP
46D4: 4C 79 47 06410          JMP FF
46D7: BD 2E 3B 06415 .22      LDA FLAGU,X
46DA: D0 10    06420          BNE .23
46DC: DE 1A 3B 06425          DEC Y,X
46DF: A9 00    06430          LDA #$00
46E1: 9D 65 3B 06435          STA DL,X
46E4: A9 01    06440          LDA #$01
46E6: 9D 6D 3B 06445          STA DU,X
46E9: 4C 79 47 06450          JMP FF
46EC: BD 2A 3B 06455 .23      LDA FLAGR,X
46EF: D0 0D    06460          BNE .24
46F1: FE 1A 3B 06465          INC Y,X
46F4: A9 00    06470          LDA #$00
46F6: 9D 65 3B 06475          STA DL,X
46F9: A9 01    06480          LDA #$01
46FB: 9D 71 3B 06485          STA DD,X
46FE: 4C 79 47 06490 .24      JMP FF
4701: BD 6D 3B 06495 .3       LDA DU,X
4704: F0 37    06500          BEQ .4
4706: BD 2E 3B 06505 .31      LDA FLAGU,X
4709: D0 06    06510          BNE .32
470B: 20 B5 45 06515          JSR AUTOP
470E: 4C 79 47 06520          JMP FF
4711: BD 26 3B 06525 .32      LDA FLAGL,X
4714: D0 10    06530          BNE .33
4716: DE 15 3B 06535          DEC X,X
```

```
4719: A9 00    06540          LDA #$00
471B: 9D 6D 3B 06545          STA DU,X
471E: A9 01    06550          LDA #$01
4720: 9D 65 3B 06555          STA DL,X
4723: 4C 79 47 06560          JMP FF
4726: BD 2A 3B 06565 .33      LDA FLAGR,X
4729: C9 01    06570          CMP #$01
472B: F0 0D    06575          BEQ .34
472D: FE 15 3B 06580          INC X,X
4730: A9 00    06585          LDA #$00
4732: 9D 6D 3B 06590          STA DU,X
4735: A9 01    06595          LDA #$01
4737: 9D 69 3B 06600          STA DR,X
473A: 4C 79 47 06605 .34      JMP FF
473D: BD 71 3B 06610 .4       LDA DD,X
4740: D0 03    06615          BNE .41
4742: 4C 79 47 06620          JMP FF
4745: BD 32 3B 06625 .41      LDA FLAGD,X
4748: D0 06    06630          BNE .42
474A: 20 B5 45 06635          JSR AUTOP
474D: 4C 79 47 06640          JMP FF
4750: BD 26 3B 06645 .42      LDA FLAGL,X
4753: D0 10    06650          BNE .43
4755: DE 15 3B 06655          DEC X,X
4758: A9 00    06660          LDA #$00
475A: 9D 71 3B 06665          STA DD,X
475D: A9 01    06670          LDA #$01
475F: 9D 65 3B 06675          STA DL,X
4762: 4C 79 47 06680          JMP FF
4765: BD 2A 3B 06685 .43      LDA FLAGR,X
4768: C9 01    06690          CMP #$01
476A: F0 0D    06695          BEQ FF
476C: FE 15 3B 06700          INC X,X
476F: A9 00    06705          LDA #$00
4771: 9D 71 3B 06710          STA DD,X
4774: A9 01    06715          LDA #$01
4776: 9D 69 3B 06720          STA DR,X
4779: 60       06725 FF       RTS
477A: AD 1E 3B 06730 PLACE    LDA XB       ;PLAYER#0 ON LEFT SIDE SCREEN?
477D: C9 0A    06735          CMP #$0A
477F: B0 5C    06740          BGE .3
4781: AD 22 3B 06745          LDA YB       ;PLAYER #0 ON TOP HALF SCREEN?
4784: C9 06    06750          CMP #$06
4786: B0 35    06755          BGE .2
4788: A9 B8    06760          LDA #$B8     ;PUT PLAYER AT BLOCK 17,7 (184,148)
478A: 99 15 3B 06765          STA X,Y
478D: A9 94    06770          LDA #$94
478F: 99 1A 3B 06775          STA Y,Y
4792: A9 00    06780          LDA #$00
4794: 99 6D 3B 06785          STA DU,Y
4797: 99 71 3B 06790          STA DD,Y
479A: 99 69 3B 06795          STA DR,Y
479D: 99 65 3B 06800          STA DL,Y
47A0: AD 0A D2 06805          LDA RANDOM   ;VARY THE START LEFT OR RT
47A3: C9 80    06810          CMP #$80
47A5: 90 0B    06815          BLT .15
47A7: A9 01    06820          LDA #$01
47A9: 99 69 3B 06825          STA DR,Y
47AC: 99 32 3B 06830          STA FLAGD,Y
47AF: 4C 30 48 06835          JMP EXIT
47B2: A9 01    06840 .15      LDA #$01
47B4: 99 65 3B 06845          STA DL,Y
```

```
47B7: 99 32 3B 06850          STA FLAGD,Y
47BA: 4C 30 48 06855          JMP EXIT
47BD: A9 A8    06860 .2       LDA #$A8     ;PUT PLAYER AT BLOCK 15,2 (168,68)
47BF: 99 15 3B 06865          STA X,Y
47C2: A9 44    06870          LDA #$44
47C4: 99 1A 3B 06875          STA Y,Y
47C7: A9 00    06880          LDA #$00
47C9: 99 6D 3B 06885          STA DU,Y
47CC: 99 69 3B 06890          STA DR,Y
47CF: 99 65 3B 06895          STA DL,Y
47D2: A9 01    06900          LDA #$01
47D4: 99 71 3B 06905          STA DD,Y
47D7: 99 2A 3B 06910          STA FLAGR,Y
47DA: 4C 30 48 06915          JMP EXIT
47DD: AD 22 3B 06920 .3       LDA YB       ;IS PLAYER ON TOP HALF SCREEN?
47E0: C9 06    06925          CMP #$06
47E2: B0 32    06930          BGE .4
47E4: A9 40    06935          LDA #$40     ;PUT PLAYER AT BLOCK 2,7 (64,148)
47E6: 99 15 3B 06940          STA X,Y
47E9: A9 94    06945          LDA #$94
47EB: 99 1A 3B 06950          STA Y,Y
47EE: A9 00    06955          LDA #$00
47F0: 99 6D 3B 06960          STA DU,Y
47F3: 99 71 3B 06965          STA DD,Y
47F6: 99 65 3B 06970          STA DL,Y
47F9: 99 69 3B 06975          STA DR,Y
47FC: AD 0A D2 06980          LDA RANDOM   ;VARY THE START LEFT OR RT
47FF: C9 80    06985          CMP #$80
4801: 90 08    06990          BLT .35
4803: A9 01    06995          LDA #$01
4805: 99 65 3B 07000          STA DL,Y
4808: 4C 30 48 07005          JMP EXIT
480B: A9 01    07010 .35      LDA #$01
480D: 99 69 3B 07015          STA DR,Y
4810: 99 32 3B 07020          STA FLAGD,Y
4813: 4C 30 48 07025          JMP EXIT
4816: A9 50    07030 .4       LDA #$50     ;PUT PLAYER AT BLOCK 4,2 (80,68)
4818: 99 15 3B 07035          STA X,Y
481B: A9 44    07040          LDA #$44
481D: 99 1A 3B 07045          STA Y,Y
4820: A9 00    07050          LDA #$00
4822: 99 65 3B 07055          STA DL,Y
4825: 99 69 3B 07060          STA DR,Y
4828: 99 6D 3B 07065          STA DU,Y
482B: A9 01    07070          LDA #$01
482D: 99 71 3B 07075          STA DD,Y
4830: 60       07080 EXIT     RTS
```

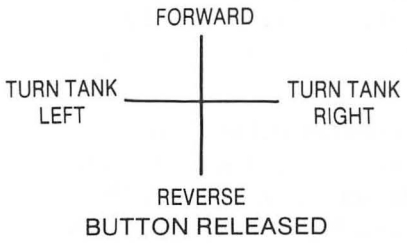
Tank Game

A tank duel between two or more tanks was one of the first few classic/games developed for coin-op play in the arcades. It was quickly translated to the Atari 2600 game system under the name *Combat*, and was supplied with the first four to five million game systems. Regrettably, it was never rewritten for their home computers. While the cartridge had some interesting variations like *Tank-Pong*, in which the shells bounced off walls for quite some distance, it was unrealistic in that the tank turret could only fire in the direction the tank pointed.

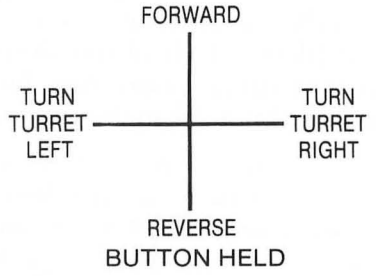
Basic Design of Game

Our game, *Tank Battle*, is designed to be much more realistic. The tanks, which can turn and drive in eight directions, are equipped with rotatable turrets so that they can fire in directions other than the one that they are traveling. The terrain features or barriers, which are useful for hiding behind, can be blown away by tank fire. It takes two shots to completely fracture one of the terrain blocks. Tanks that are blown away are immune to enemy fire for several seconds upon reappearance. This is necessary since the tank always reappears in the same position and it would be only fair to allow the player to move his tank or fire back at a tank waiting in ambush.

Regrettably, the control system is a compromise. Game design would be so much easier if joysticks had two buttons. The single button serves a dual function for turning the turret and firing the gun. When the button isn't pressed, the tank is in the steering-drive mode. Pushing to the left or right rotates the tank, and pushing forward or back moves the tank in those directions. Since the tank can't penetrate barriers, it often needs to be backed up to turn away from the obstacle. Pressing the button puts it into the turret-fire mode. The tank can still be driven forward and backward, but pushing the stick left or right rotates the turret counterclockwise and clockwise respectively. The gun is fired when the button is released.



BUTTON RELEASED



BUTTON HELD

The use of two players for each tank, one for the body and one for the turret, thwarted any attempts to make it a four player game. We're not saying that it couldn't be done, but we could just see the flickering that would occur if two tanks using the same players ended up on the same scan lines. Actually, now that we think

about it, it could have been done if we sacrificed the distinct second color of the turret. The turret might not show clearly, but it would reduce each complete tank to one player. Now you would need sixty-four tank shapes, one for each combination of tank direction and turret direction. As it is we use eight tank shapes and eight separate turret shapes for each of our eight joystick-controlled directions.

Only one missile can appear on the screen at a time for each tank. Each missile continues along its directed path until it exits the playfield or reaches either the enemy tank or a blow-away wall. Since the missile is armed upon pressing the trigger, it becomes increasingly important to be able to not only adjust the turret's direction but also the tank's forward and backward position before the shell is actually fired by releasing the trigger. The inablity to reload and fire before the previous missile completes its trajectory may be quite realistic in tank warfare, but can be a liability in an arcade game. Perhaps this forces the game to transcend the quick reflexes genre to one of strategic play.

The game was virtually designed to run in Deferred Vertical Blank. While collisions, explosions, and missile movement are performed for both tanks every VBlank cycle, reading the joystick and the actual calculation and updating of tank and turret positions alternate for each tank every VBlank cycle. Tank #0 is updated on even cycles and tank #1 is updated on odd cycles. This was necessary for it appeared that there were far too many instructions to fit within the Deferred Vertical Blank period.

Updating the tank's position is one of the more intriguing problems in this game. Calculating the tank's next position based on the direction that it is moving is quite straightforward. The formulas are:

```
PLAYERH = PLAYERH + HOFFS(NEWD)
PLAYERV = PLAYERV + VOFFS(NEWD)
```

where NEWD is the tank's new direction.

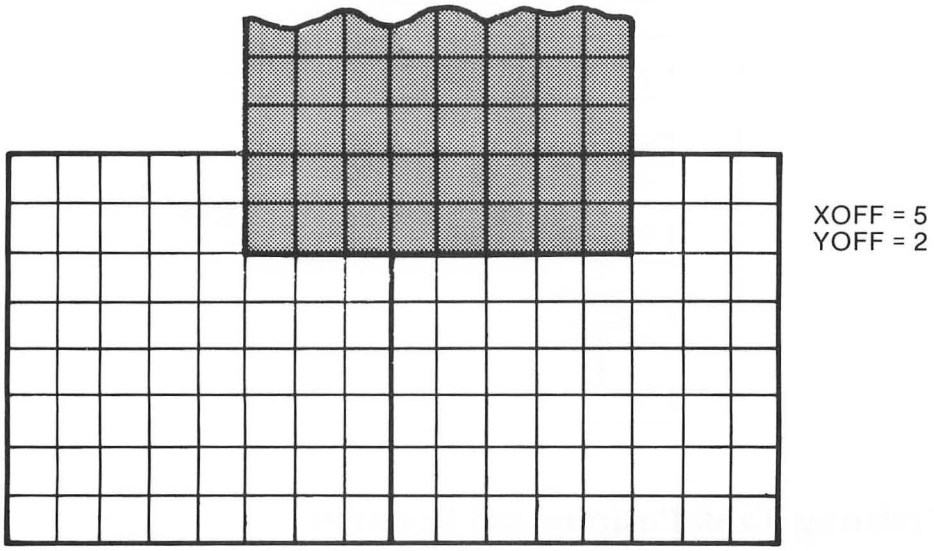
Tank Collision with Playfield

The tank obviously has to be prevented from moving into a wall. We wanted to avoid the unsightly bounce that occurs in many games that test illegal positions using player-playfield collisions. The problem is that you can't test the collision until you actually move there, but once you are there you have to move back from the illegal position, or in this case, away from the wall.

The solution is to test for a collision at the tank's next calculated position before the tank is actually moved there. This can be accomplished exactly the same way collisions are detected when rastering shapes on the screen, by ANDing the player byte against the playfield byte beneath it. If any two bits overlap, the result is a positive number. Remember that to do the comparison, the tank doesn't have to be physically where it is supposed to be. You need only calculate which player and playfield character bytes intersect and compare the data for an overlap.

By calculating the tank's new position NEWH, NEWV, the offset into the character XOFF, YOFF can be determined simply by ANDing the position with #$07. This is equivalent to the expression NEWH -8 (INT(NEWH/8)). It masks off the higher bits above 8 so that only the remainder is left. The actual character involved in the intersection also has to be obtained. Its position is at CHRX, CHRY and is obtained by dividing the new player position NEWH, NEWV by 8 in each axis. Its position in memory is CHRHI*256 + (CHRX +(20*CHRY)).

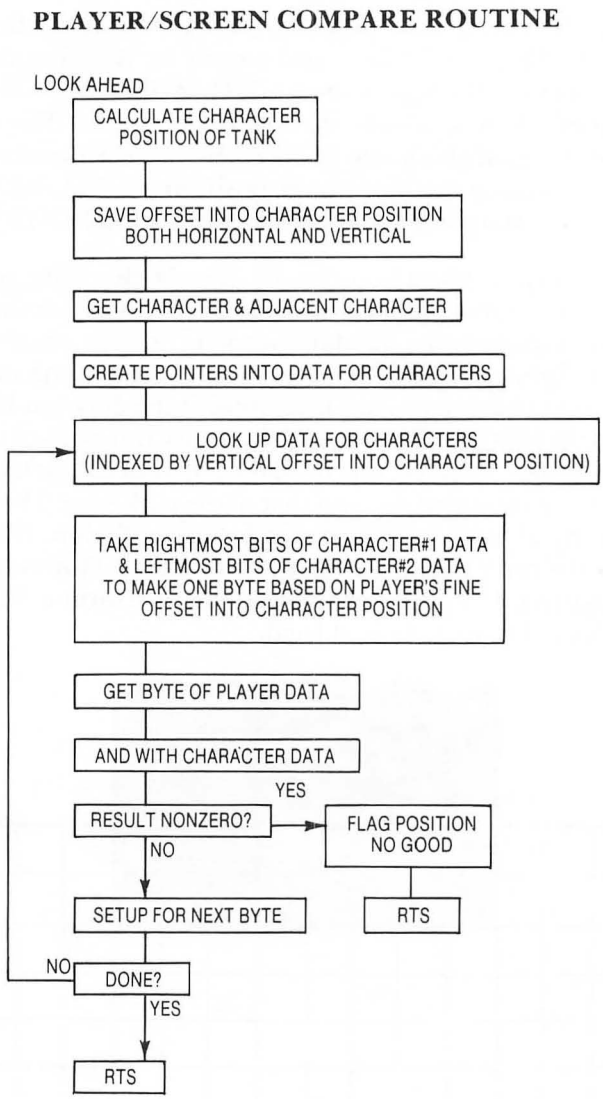
When the player shape is offset horizontally into the character, part of its shape is in that character and the rest extends into the next character to the right. If we are going to make a comparison in the data, we will need to obtain data from that character also. The eight bits that we need to compare are beneath the player byte but span the two different character bytes. If we look at the diagram below, our player shape is offset into the first character by five bits. Thus, three bits of the first character byte need to be combined with five bits from the adjacent character. Once we have a byte we can AND the character data with that of the tank data. The data is tested for each byte of the vertical overlap or until we detect a collision. If we don't detect a collision, we clear the carry before exiting the subroutine. However, if we do, we set the carry before exiting. Upon return we need only perform a BCS instruction to determine if the desired move is indeed legal.



We intersect by 2 bytes

0th byte overlapped character data $FFintersects tank data $38
1st byte overlapped character data $FF intersects tank data $C6

LDA (P0TMP1),Y	$38	1 1 0 0 0 1 1 0
AND BYTE	$FF	1 1 1 1 1 1 1 1
RESULT >0 (Collision)	$38	1 1 0 0 0 1 1 0



Updating Tank Position and Rotation

The two tanks are updated on different VBlank cycles. While it is true that tank #0 is updated on even cycles and tank #1 is updated on odd cycles, there is a rest of two additional cycles before they are updated again. Essentially, we are in a four-cycle loop with two null cycles. If you look at the code beginning with the label UPDATE, you will notice that we loaded the Accumulator with the clock timer value and then ANDed it with #$02. If you look at the bit patterns for the numbers 0-3 and perform the operation, you will get the following:

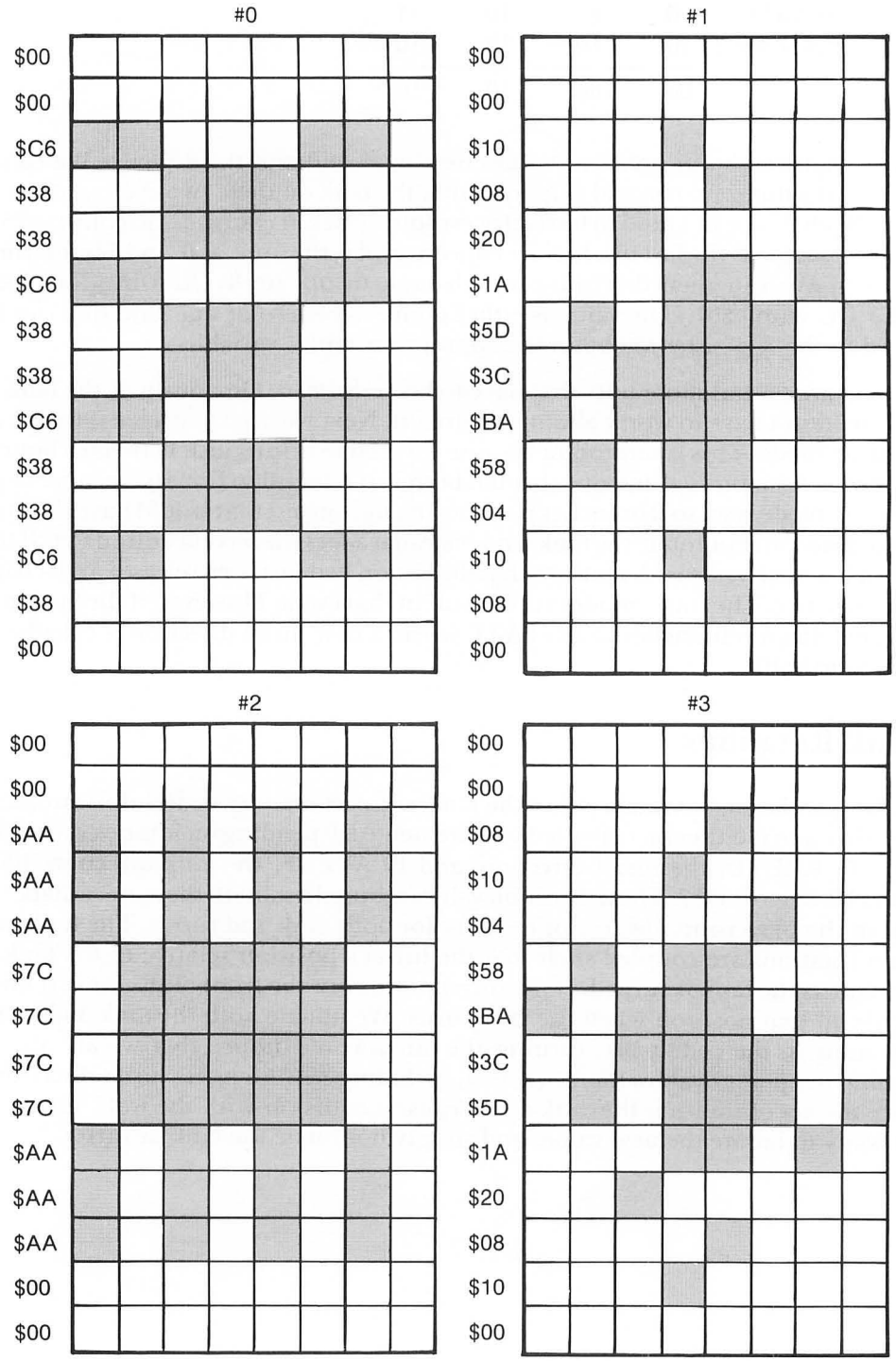
ACCUM.	00	01	10	11
AND #$02	10	10	10	10
	00	00	10	10

Notice that we obtain non-zero values for the second and third cycles. We could branch on a non-zero test and skip updating the tank on these two cycles. Incidentally, the clock doesn't need to be reset every four cycles. It continues to count to 255. However, as far as the last two bits are concerned, 4 is the same as 0, and 5 is the same as 1, etc. We can then determine which tank to update by ANDing the clock RTCLOC with #$01. The value is either going to be zero or one, and that can be placed in the X-register to obtain the appropriate tank's variables.

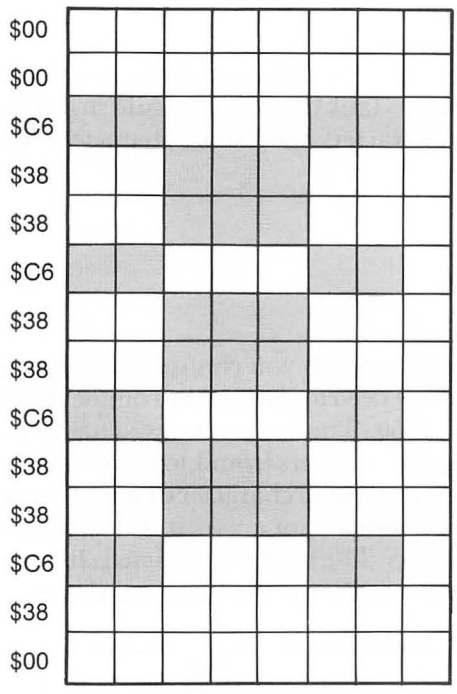
There are several more tests that have to be performed. Obviously, if the tank is dead, we don't have to worry about updating it. Next we need to decide if we are in the turret mode. This is important because we need to distinguish between a button that hasn't been pressed and one that has been just released to fire. In the latter case, the turret mode is set so it branches past the first button test that would have shunted it to code to move or rotate the tank, and instead reaches the second button test. If the button is actually released, it means that the button had just been released and it trips the gun to fire. The turret mode is turned off in that event. However, if the button is still held down it branches to ROTATE where a new turret direction is calculated every fourth jiffy.

Tank Rotations

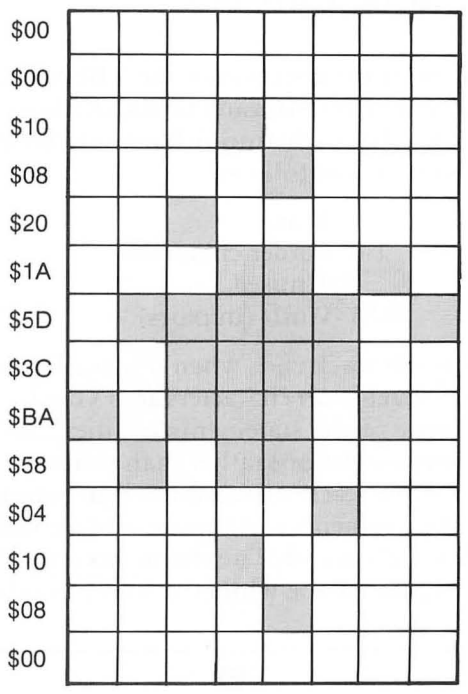
The calculations to rotate either the tank or turret are very straightforward. The new direction is either incremented or decremented depending on joystick direction. Both TURRETD, the turret direction, and PLAYERF, the tank direction, have values between 0 and 7. These direction values are used to obtain the correct shapes to plot in the player-missile area of memory for both tank and turret. The tank and turret rotations are coupled such that the turret's position relative to the tank is constant as the tank is turned. If the turret points out the front of the tank, it must remain in that position when the tank turns. We update both the tank and turret directions in the code when turning the tank. You'll notice that we are always keeping temporary values for movement, including new tank and turret directions. Since the act of rotating the tank could cause a collision with the wall, it may be necessary to ignore the new values and simply not rotate the tank or turret.


#0
$00
$00
$C6
$38
$38
$C6
$38
$38
$C6
$38
$38
$C6
$38
$00
#1
$00
$00
$10
$08
$20
$1A
$5D
$3C
$BA
$58
$04
$10
$08
$00
#2
$00
$00
$AA
$AA
$AA
$7C
$7C
$7C
$7C
$AA
$AA
$AA
$00
$00
#3
$00
$00
$08
$10
$04
$58
$BA
$3C
$5D
$1A
$20
$08
$10
$00

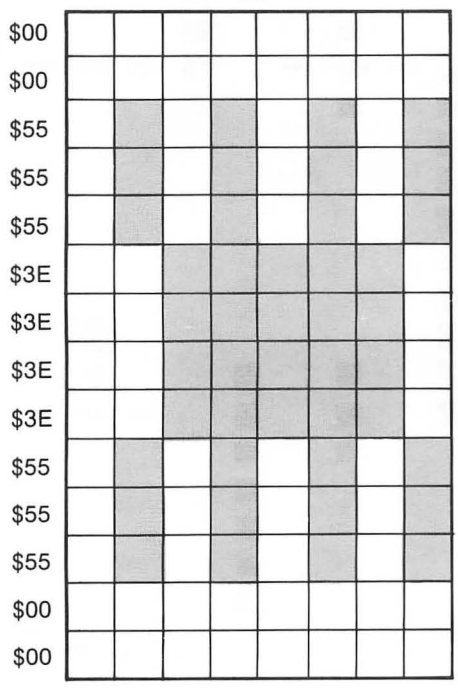
#4


$00
$00
$C6
$38
$38
$C6
$38
$38
$C6
$38
$38
$C6
$38
$00

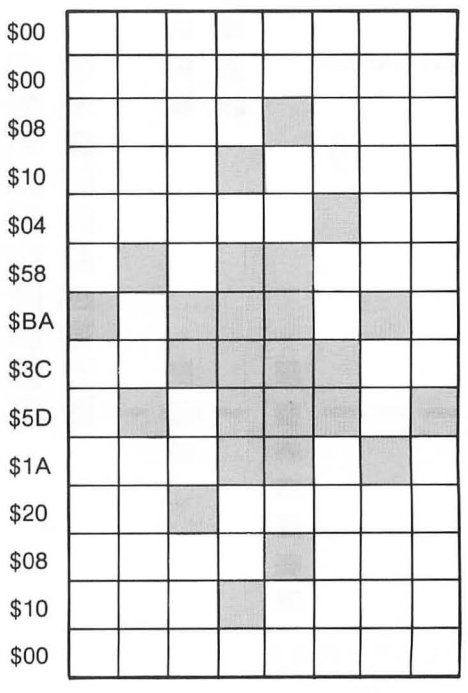
#5


$00
$00
$10
$08
$20
$1A
$5D
$3C
$BA
$58
$04
$10
$08
$00

#6


$00
$00
$55
$55
$55
$3E
$3E
$3E
$3E
$55
$55
$55
$00
$00

#7

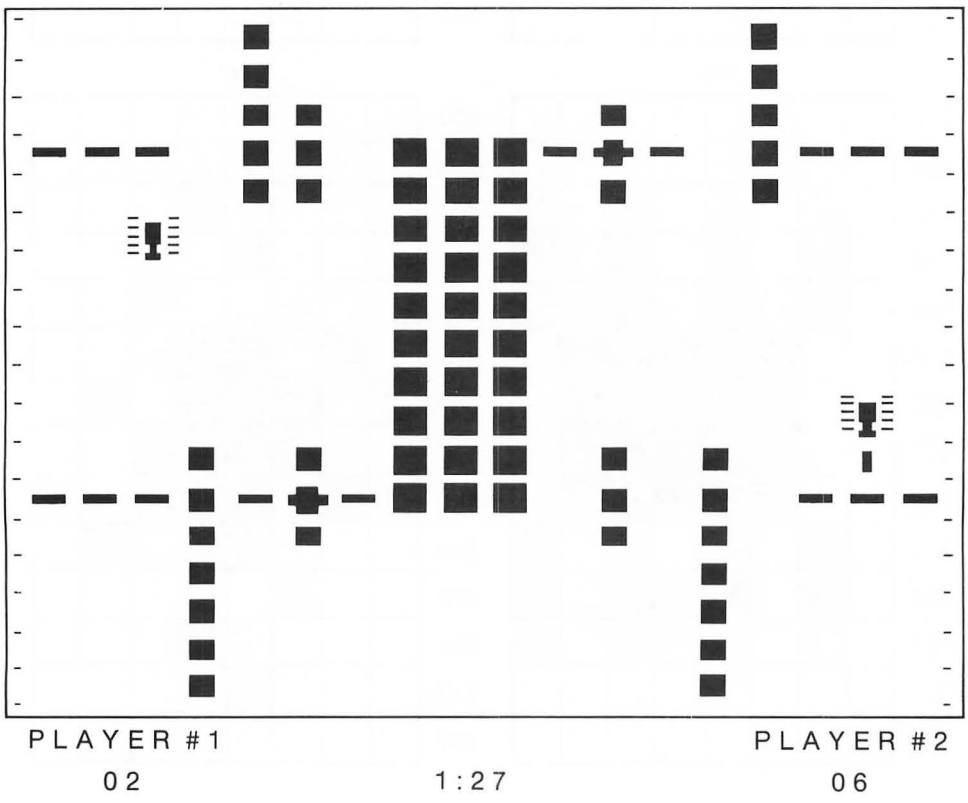

$00
$00
$08
$10
$04
$58
$BA
$3C
$5D
$1A
$20
$08
$10
$00

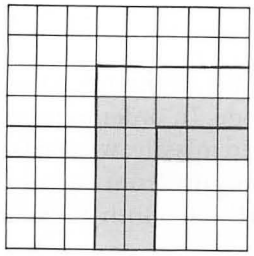
Missile or Tank Fire

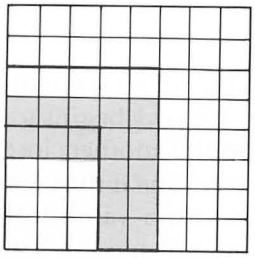
The beginning section of the VBlank code concerns collisions between missiles and playfield, missiles and tanks, and between two tanks. When a missile strikes the playfield it has to distinquish between border characters and barrier characters. The character set is as follows:

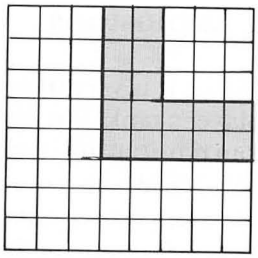
0 -Blank
1-6 -Border characters
7 -Unused
8-15 -Walls (in pairs)

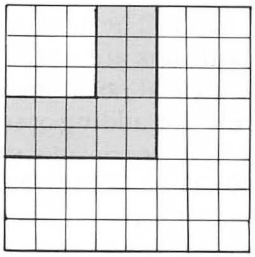
The missile track ends when it hits a border character. If you choose to change the maze so that border characters aren't used, you will need to remove the comment field from some of the statements in lines 2540 to 2760. These statements actually test boundary conditions rather than simply character numbers from 1 to 6 to determine if the missile reaches the border. When the missile strikes a character that makes up a wall, it decrements it if it is an odd numbered character, or erases it if it is an even numbered character. The characters are set in pairs. The higher numbered character is a complete shape while the lower numbered character is the fractured shape.

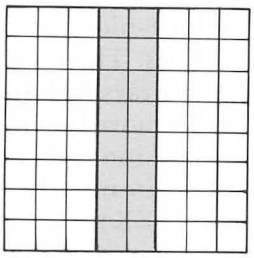


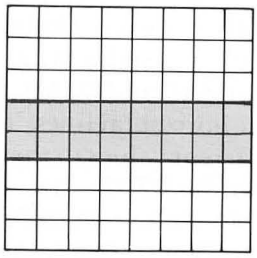

#1

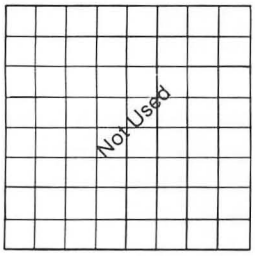

#2


#3

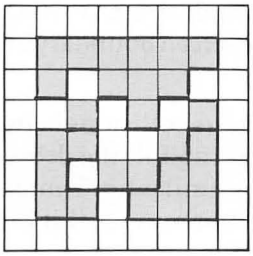

#4

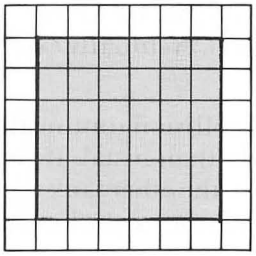

#5

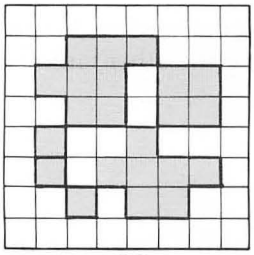

#6

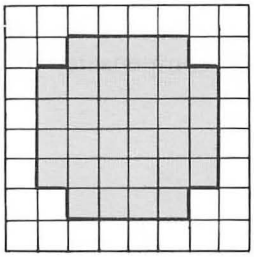


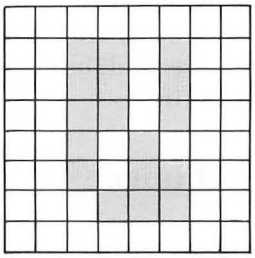
#7

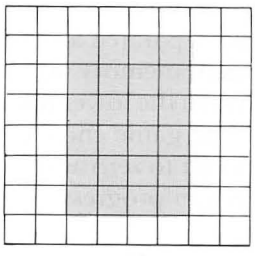

#8

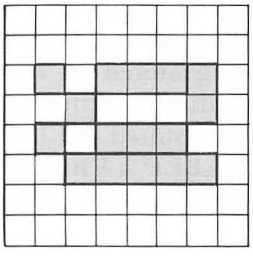

#9

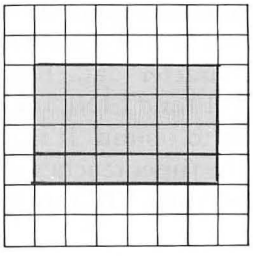

#10


#11


#12


#13


#14


#15

Tank Explosions

Of course, it is very likely that the player's missile will hit its intended target, the other player's tank. If it does, the hit tank begins to explode. In order to prevent the tank from reinitiating its explosion by another closely fired missile, we need to test if CYCLES >0. We also need to determine if the tank is immune from enemy fire just after it reappears. Again we test for a value IMMUNE >0. Both CYCLES and IMMUNE are counters that count down every VBlank.

The tank's explosion pattern is a series of horizontal lines that rapidly expand vertically outward along the tank's 8-bit-wide player band. The tank shape is replaced by a series of horizontal lines. Their vertical positions are stored in two arrays, DZAP and UZAP. These two arrays each store the vertical positions of eight of these horizontal lines. The lines are moved at different rates by adding or subtracting different values to their vertical positions. For example, if we consider the eight lines in the DZAP array, they all begin initally at X=80. Then DZAP(INDEX) = DZAP(INDEX) + INDEX, where INDEX ranges from 0 to 7 for the eight values in the table. The higher values in the table move the fastest downward. Eventually all of the lines exceed the screen boundary at #$C0 and the explosion ends. Similarly, the eight lines in UZAP move upward as UZAP(INDEX) = UZAP(INDEX) -INDEX. They, too, eventually exceed the screen boundary at #$20 and the explosion sequence ends.

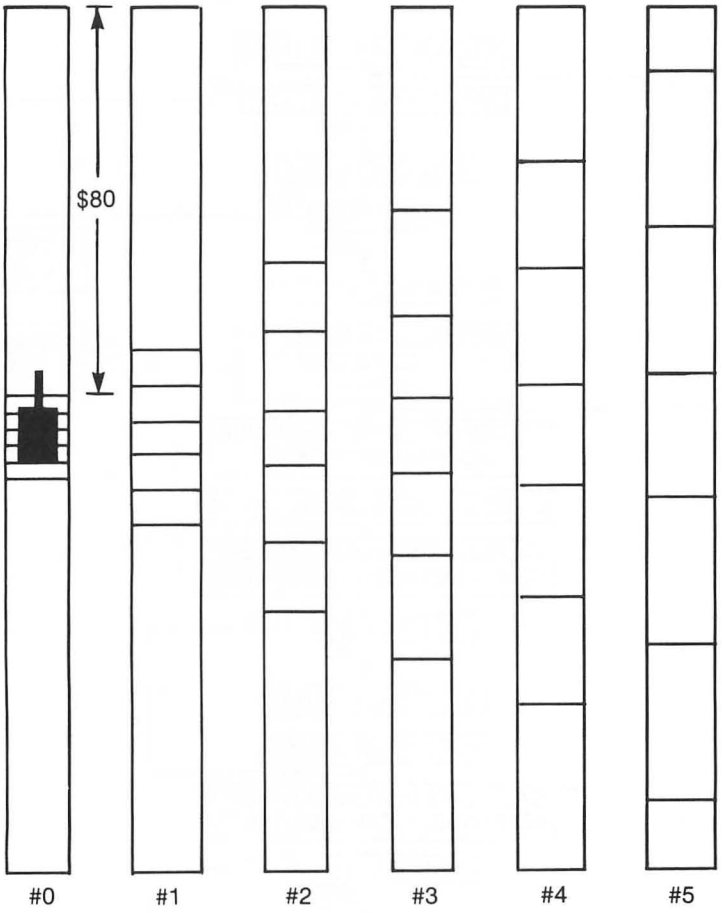
The collision routine that detects collisions between the two tanks has to be more selective than usual. If we aren't careful, the debris from the exploding tank could blow up the other tank if it had a similar horizontal position. This can be avoided by checking the explosion cycle counter to see if it is greater than zero.

The blown up tank is put back on the screen once the CYCLE counter runs out after four seconds. The IMMUNITY counter is then set to 255 jiffies or four seconds so that the enemy tank can't sit in ambush. It does give the defeated tank a slight edge for several seconds but this can be compensated by having the other tank stay out of the line of fire.

Game Timer

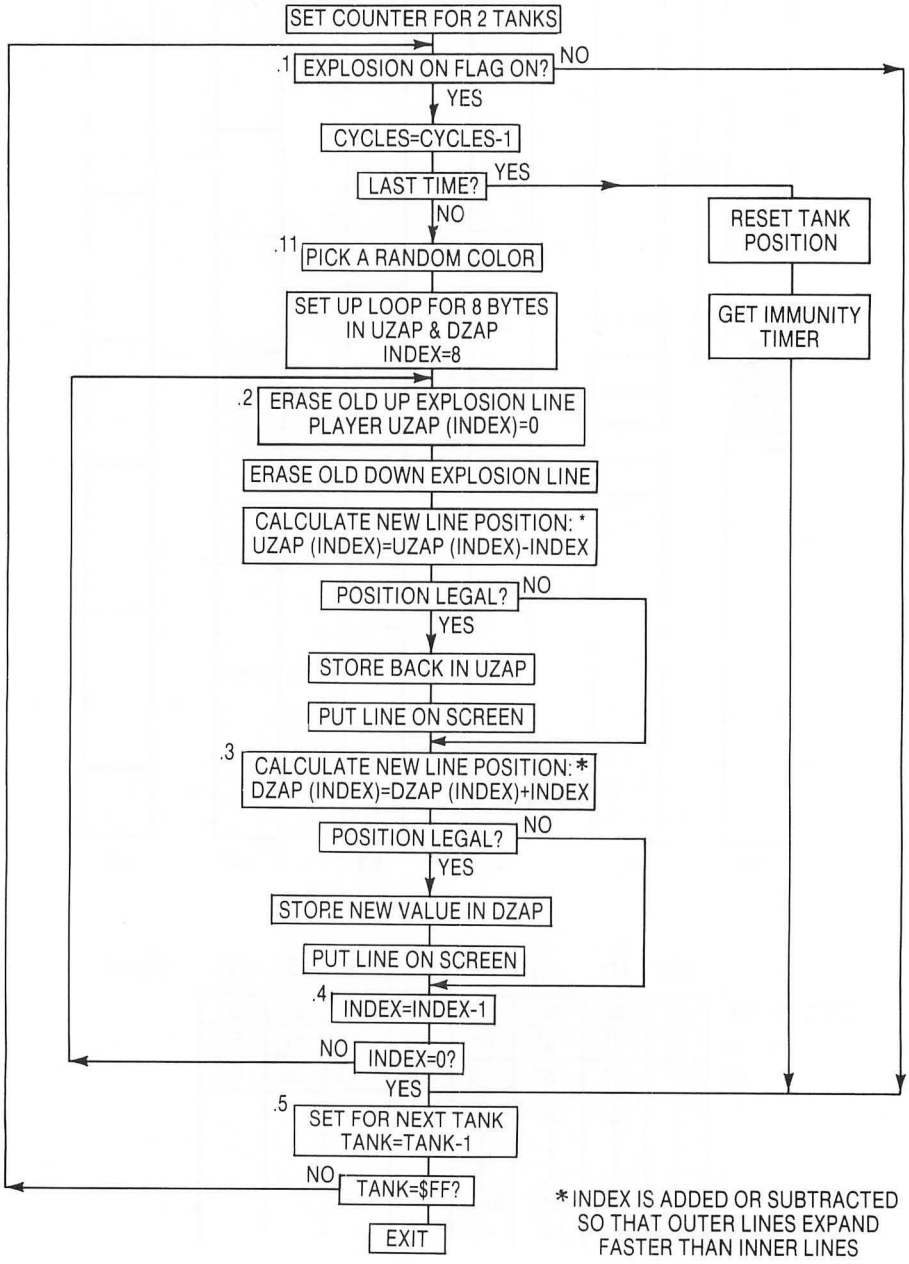
The game is timed to last three minutes. This is set up initally as two minutes and sixty seconds, and can be changed to suit the player. The timer operates in the decimal mode. Both the ones and tens digits for seconds are stored in the lower and upper nibbles of the variable SECONDS. The nibbles are separated and converted into character data before being stored in score screen memory at locations LINE3+10 and LINE3+11. The value for minutes, which is in the lower nibble only, is easier to obtain. It is eventually stored at LINE3+8. The game ends, GAME=0, when the timer reaches zero. ENABLE, a VBI flag, is also set to zero at that time so that the entire VBI routine is skipped when the game isn't in progress.

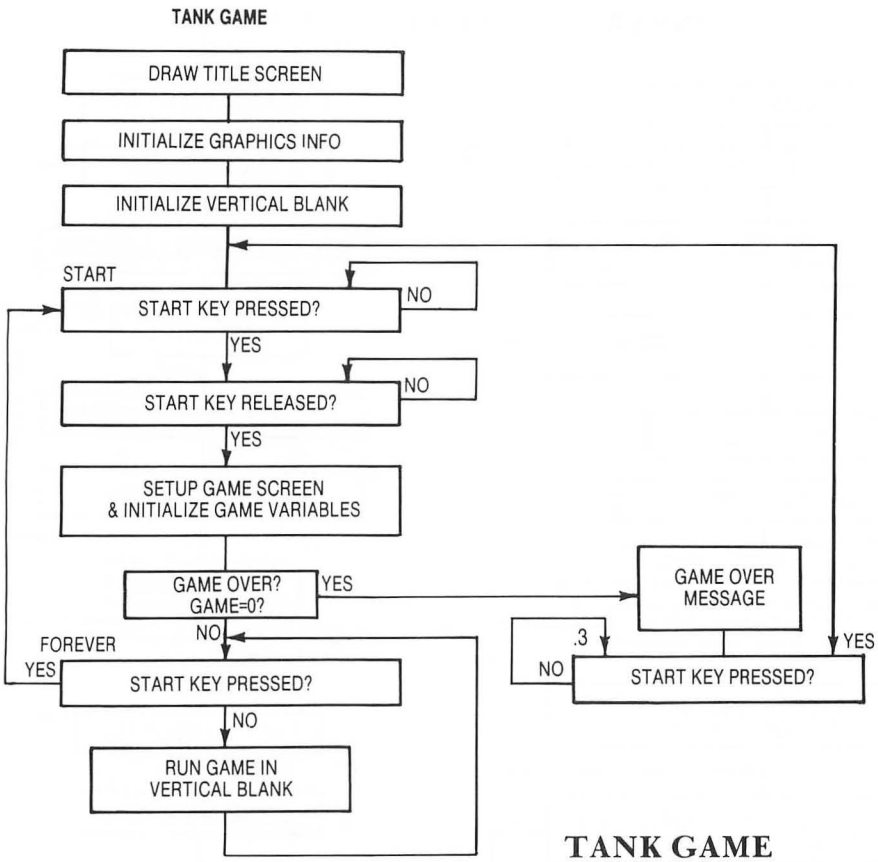
EXPLOSION PATTERN



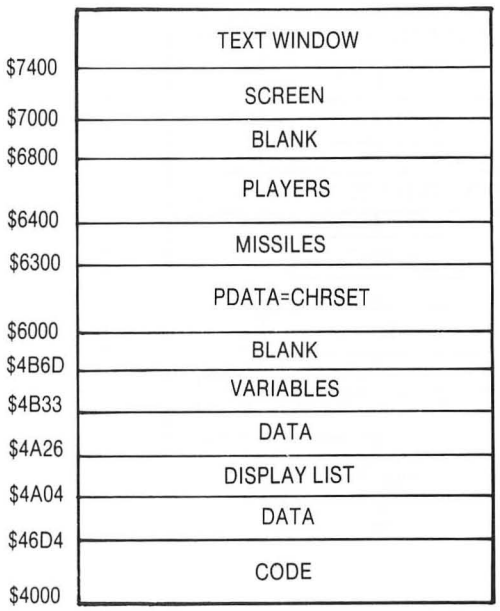
	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	DZAP
CYCLE #0	80	80	80	80	80	80	80	80	
#1	81	82	83	84	85	86	87	88	
#2	82	34	86	88	8A	8C	8E	90	
	↑ Add +1	↑ Add +2	↑ Add +3	↑ Add +4	etc.				

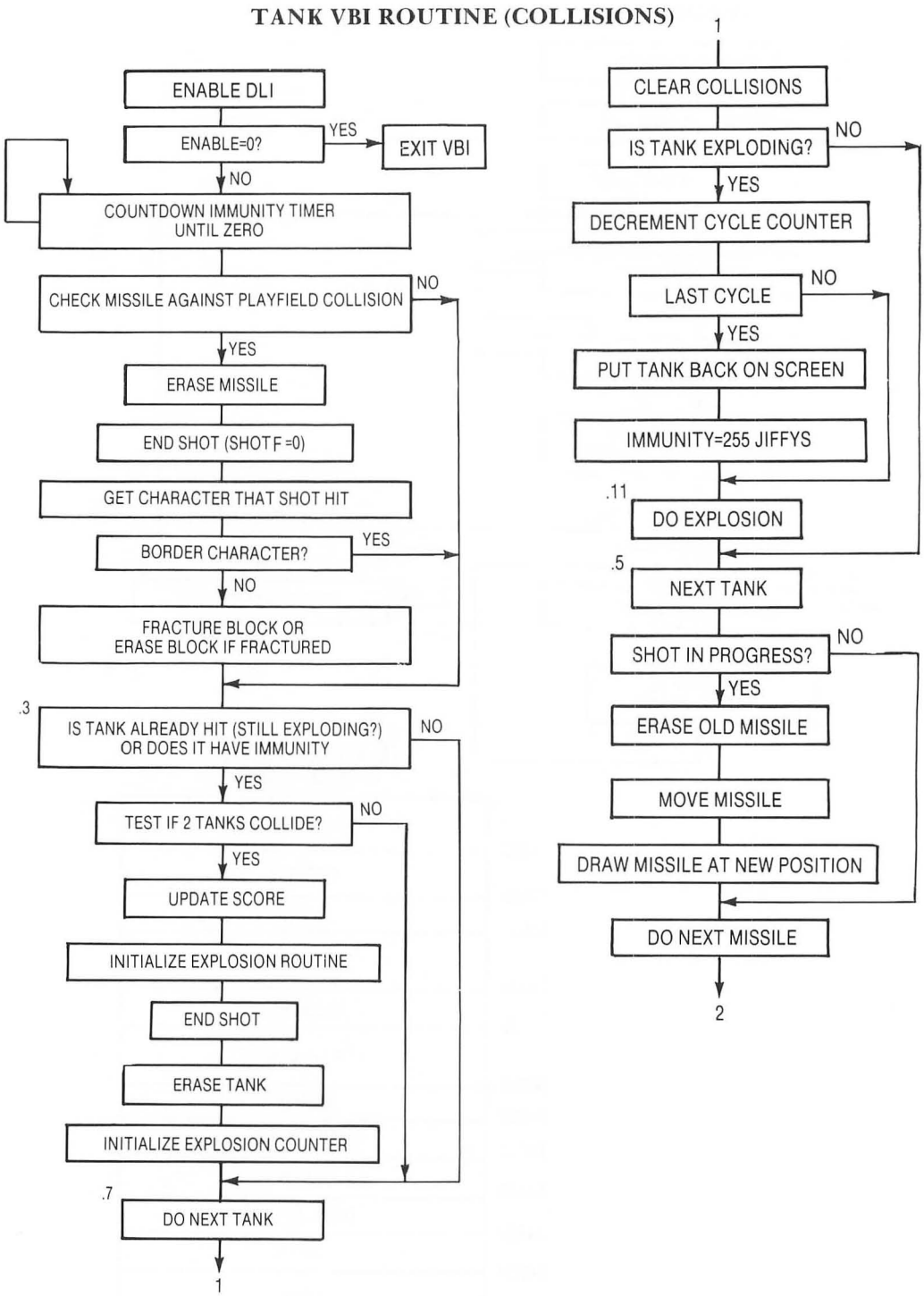
TANK EXPLOSION SUBROUTINE

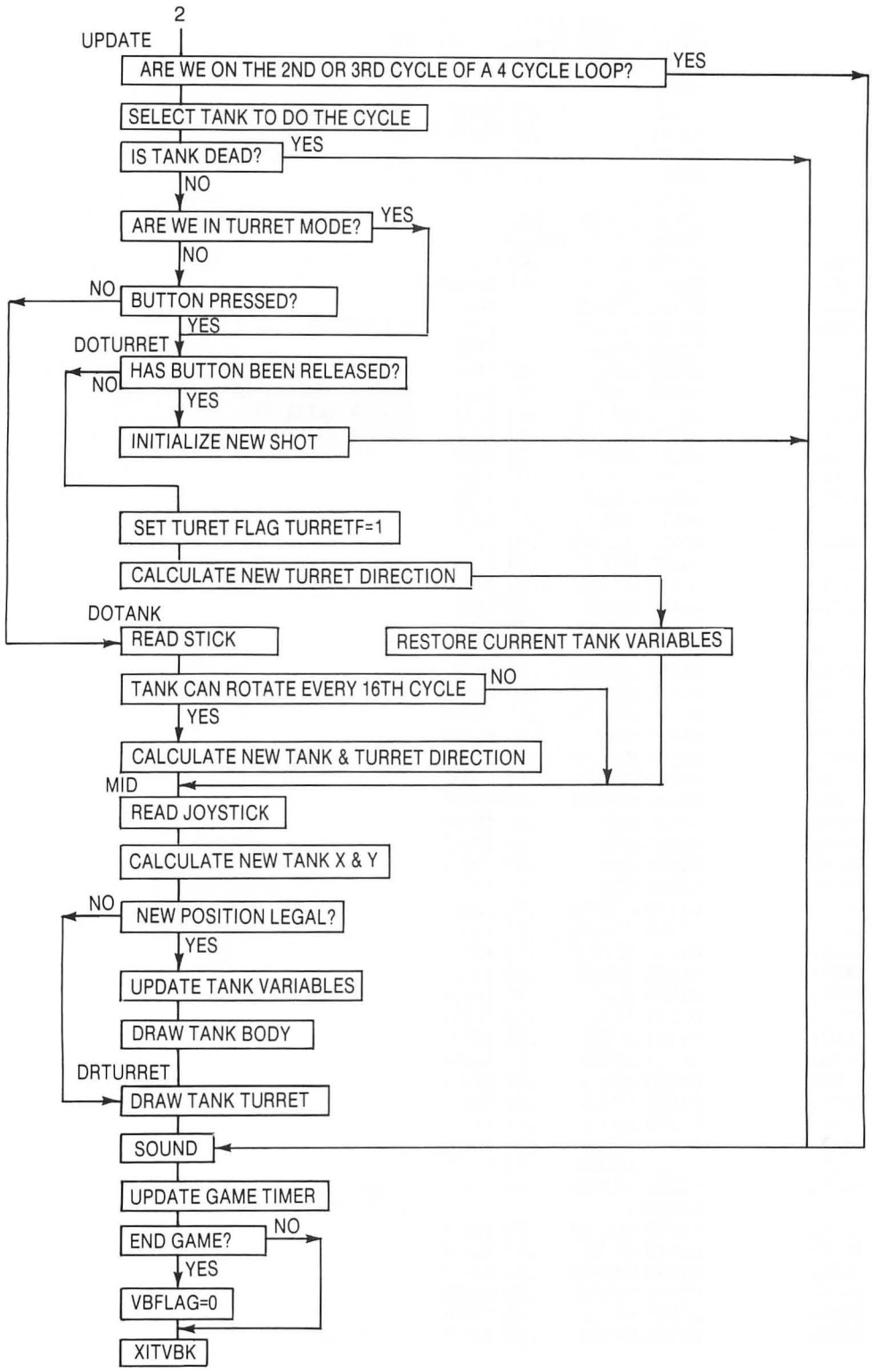




TANK GAME
MEMORY MAP




TANK VBI ROUTINE (COLLISIONS)
ENABLE DLI
ENABLE=0?
YES
EXIT VBI
NO
COUNTDOWN IMMUNITY TIMER UNTIL ZERO
CHECK MISSILE AGAINST PLAYFIELD COLLISION
NO
YES
ERASE MISSILE
END SHOT (SHOTF=0)
GET CHARACTER THAT SHOT HIT
BORDER CHARACTER?
YES
NO
FRACTURE BLOCK OR ERASE BLOCK IF FRACTURED
.3
IS TANK ALREADY HIT (STILL EXPLODING?) OR DOES IT HAVE IMMUNITY
NO
YES
TEST IF 2 TANKS COLLIDE?
NO
YES
UPDATE SCORE
INITIALIZE EXPLOSION ROUTINE
END SHOT
ERASE TANK
INITIALIZE EXPLOSION COUNTER
.7
DO NEXT TANK
1
1
CLEAR COLLISIONS
IS TANK EXPLODING?
NO
YES
DECREMENT CYCLE COUNTER
LAST CYCLE
NO
YES
PUT TANK BACK ON SCREEN
IMMUNITY=255 JIFFYS
.11
DO EXPLOSION
.5
NEXT TANK
SHOT IN PROGRESS?
NO
YES
ERASE OLD MISSILE
MOVE MISSILE
DRAW MISSILE AT NEW POSITION
DO NEXT MISSILE
2


2
UPDATE
ARE WE ON THE 2ND OR 3RD CYCLE OF A 4 CYCLE LOOP?
YES
SELECT TANK TO DO THE CYCLE
IS TANK DEAD?
YES
NO
ARE WE IN TURRET MODE?
YES
NO
NO
BUTTON PRESSED?
YES
DOTURRET
HAS BUTTON BEEN RELEASED?
NO
YES
INITIALIZE NEW SHOT
SET TURET FLAG TURRETF=1
CALCULATE NEW TURRET DIRECTION
DOTANK
READ STICK
RESTORE CURRENT TANK VARIABLES
TANK CAN ROTATE EVERY 16TH CYCLE
NO
YES
CALCULATE NEW TANK & TURRET DIRECTION
MID
READ JOYSTICK
CALCULATE NEW TANK X & Y
NO
NEW POSITION LEGAL?
YES
UPDATE TANK VARIABLES
DRAW TANK BODY
DRTURRET
DRAW TANK TURRET
SOUND
UPDATE GAME TIMER
END GAME?
NO
YES
VBFLAG=0
XITVBK

```
                00010 * TANK BATTLE - COPYRIGHT 1984 BY DAN PINAL
                00030 * TANK GAME - MASTER FILE
                00040 * THIS IS D:TANKMAST.SRC
                00050 *
                00060          .OR $4000
                00070 *        .TF "D:TANK.OBJ"
                00080 *
                00090          .IN "D:EQUATES"
                00010 ;
                00020 ; EQUATE FILE
                00030 ; O.S. EQUATES
0012:           00070 RTCLOK   .EQ $12
0014:           00080 RTCLOC   .EQ RTCLOK+2
004D:           00090 ATRACT   .EQ $4D
0200:           00100 VDSLST   .EQ $200    DLI VECTOR
0224:           00110 VVBLKD   .EQ $224
022F:           00120 SDMCTL   .EQ $22F    SHADOW DMACTL
0230:           00130 SDLSTL   .EQ $230    SHADOW DISPLAY LIST POINTER
0231:           00140 SDLSTH   .EQ $231    SHADOW DLIST HI
026F:           00150 GPRIOR   .EQ $26F    SHADOW PRIORITY REG.
0278:           00240 STICK0   .EQ $278
0284:           00360 STRIG0   .EQ $284
02C0:           00400 PCOLR0   .EQ $2C0
02C1:           00410 PCOLR1   .EQ $2C1
02C2:           00420 PCOLR2   .EQ $2C2
02C3:           00430 PCOLR3   .EQ $2C3
02C4:           00440 COLOR0   .EQ $2C4
02C5:           00450 COLOR1   .EQ $2C5
02C6:           00460 COLOR2   .EQ $2C6
02C7:           00470 COLOR3   .EQ $2C7
02C8:           00480 COLOR4   .EQ $2C8
02F4:           00490 CHBAS    .EQ $2F4    SHADOW CHR BAS
D000:           00750 HPOSP0   .EQ $D000
D000:           00760 M0PF     .EQ $D000
D001:           00770 HPOSP1   .EQ $D001
D002:           00790 HPOSP2   .EQ $D002
D004:           00830 HPOSM0   .EQ $D004
D004:           00840 P0PF     .EQ $D004
D005:           00850 HPOSM1   .EQ $D005
D008:           00920 M0PL     .EQ $D008
D00E:           01040 P2PL     .EQ $D00E
D014:           01150 COLPM2   .EQ $D014
D01D:           01240 GRACTL   .EQ $D01D
D01E:           01250 HITCLR   .EQ $D01E
D01F:           01260 CONSOL   .EQ $D01F
D200:           01280 AUDF1    .EQ $D200
D201:           01290 AUDC1    .EQ $D201
D202:           01300 AUDF2    .EQ $D202
D203:           01310 AUDC2    .EQ $D203
D204:           01320 AUDF3    .EQ $D204
D205:           01330 AUDC3    .EQ $D205
D206:           01340 AUDF4    .EQ $D206
D207:           01350 AUDC4    .EQ $D207
D208:           01360 AUDCTL   .EQ $D208
D209:           01370 STIMER   .EQ $D209   ; (W) START TIMER (RESET AUD-FREQ DIVIDERS-
                01380 ;                    ; -TO AUDF VALUES)
D20A:           01390 RANDOM   .EQ $D20A
D20F:           01410 SKCTL    .EQ $D20F
D407:           01490 PMBASE   .EQ $D407
D409:           01500 CHBASE   .EQ $D409
D40A:           01510 WSYNC    .EQ $D40A
D40E:           01530 NMIEN    .EQ $D40E   ; FOR DLI
```

```
E45C:           01580 SETVBV   .EQ $E45C
E462:           01600 XITVBV   .EQ $E462
                00100          .IN "D:TANK.EQU"
                00010 * GAME EQUATES
                00020 * THIS IS TANK.EQU
                00030 ;
                00040 ;
00F0:           00050 CHRLO    .EQ $F0
00F1:           00060 CHRHI    .EQ $F1
00F2:           00070 DATAP1   .EQ $F2
00F4:           00080 DATAP2   .EQ $F4
00F6:           00090 POTMP0   .EQ $F6
00F7:           00100 POTMP1   .EQ $F7
00F8:           00110 POTMP2   .EQ $F8
00F9:           00120 POTMP3   .EQ $F9
00FA:           00130 POTMP4   .EQ $FA
00FB:           00140 POTMP5   .EQ $FB
00FC:           00150 POTMP6   .EQ $FC
                00160 ;
                00170 ;
6000:           00180 PDATA    .EQ $6000
6000:           00190 CHRSET   .EQ PDATA
6300:           00200 MISSILES .EQ PDATA+$300
6400:           00210 PLAYER0  .EQ MISSILES+$100
6500:           00220 PLAYER1  .EQ PLAYER0+$100
6600:           00230 PLAYER2  .EQ PLAYER1+$100
6700:           00240 PLAYER3  .EQ PLAYER2+$100
7000:           00250 SCREEN   .EQ $7000
7190:           00260 WINDOW   .EQ SCREEN+400
7190:           00270 LINE1    .EQ WINDOW
71A4:           00280 LINE2    .EQ LINE1+20
71B8:           00290 LINE3    .EQ LINE2+20
71CC:           00300 LINE4    .EQ LINE3+20
E000:           00310 ATARI    .EQ $E000   ; INTERNAL CHARACTER SET
                00320 ;
                00110          .IN "D:TANK.SRC"
                00010 ; THIS IS D:TANK.SRC
                00020 * SET UP SCREEN & PMG
                00030 TITLE
                00040 ; SET UP COLORS
4000: A2 09     00050          LDX #$09
                00060 .0
4002: BD 6C 4A  00070          LDA COLORS,X
4005: 9D C0 02  00080          STA PCOLR0,X
4008: CA        00090          DEX
4009: 10 F7     00100          BPL .0
400B: A9 07     00110          LDA #TDLIST ; PUT UP TITLE
400D: 8D 30 02  00120          STA SDLSTL
4010: A9 4B     00130          LDA /TDLIST
4012: 8D 31 02  00140          STA SDLSTH
4015: A2 00     00150          LDX #$00
4017: A0 02     00160          LDY #$02
                00170 .1
4019: A5 14     00180          LDA RTCLOC
                00190 .2
401B: C5 14     00200          CMP RTCLOC
401D: F0 FC     00210          BEQ .2
401F: CA        00220          DEX
4020: D0 F7     00230          BNE .1
4022: 88        00240          DEY
4023: D0 F4     00250          BNE .1
                00260 SETUP
```

```
4025: A9 00    00270          LDA #$00
4027: 8D 08 D2 00280          STA AUDCTL  ; INSURE CLEAN SOUND
402A: A9 03    00290          LDA #$03
402C: 8D 0F D2 00300          STA SKCTL   ; CLEAR ANY GARBAGE SOUND
402F: A9 3E    00310          LDA #$3E    ; SINGLE LINE RES. PLAYERS
4031: 8D 2F 02 00320          STA SDMCTL
4034: A9 03    00330          LDA #$03
4036: 8D 1D D0 00340          STA GRACTL
4039: A9 60    00350          LDA /PDATA
403B: 8D 07 D4 00360          STA PMBASE
403E: A9 11    00370          LDA #$11    ; MISSILE COLOR=COLPF3,PLAYER PRIORITY
4040: 8D 6F 02 00380          STA GPRIOR
4043: A9 2A    00390          LDA #NDLIST
4045: 8D 30 02 00400          STA SDLSTL
4048: A9 4A    00410          LDA /NDLIST
404A: 8D 31 02 00420          STA SDLSTH
404D: A9 00    00430          LDA #$00
404F: 8D 69 4B 00440          STA GAME    ; GAME IN PROGRESS FLAG
4052: 8D 6A 4B 00450          STA ENABLE  ; PAUSE FLAG FOR OUR VERTICAL BLANK
4055: A9 07    00460          LDA #$07    ; DEFFERED VBI
4057: A2 40    00470          LDX /VBI
4059: A0 BA    00480          LDY #VBI
405B: 20 5C E4 00490          JSR SETVBV  ; INIT OUR VERTICAL BLANK.
               00500 ; SET CHR BASE              \IT WILL INIT DLI
405E: A2 00    00510          LDX #$00
               00520 .2
4060: BD FA 46 00530          LDA NEWSET,X
4063: 9D 00 60 00540          STA CHRSET,X
4066: CA       00550          DEX
4067: D0 F7    00560          BNE .2
4069: 20 B7 45 00570          JSR DOSCREEN
406C: A9 60    00580          LDA /CHRSET
406E: 8D F4 02 00590          STA CHBAS
               00600 START
4071: A9 00    00610          LDA #$00
4073: 8D 6A 4B 00620          STA ENABLE
4076: 8D 69 4B 00630          STA GAME
4079: A9 06    00640          LDA #$06
407B: CD 1F D0 00650          CMP CONSOL  ; START PRESSED?
407E: D0 F1    00660          BNE START
4080: A9 07    00670          LDA #$07
               00680 .1
4082: CD 1F D0 00690          CMP CONSOL  ; RELEASED?
4085: D0 FB    00700          BNE .1
4087: 20 36 45 00710          JSR RESTART ; WAIT TILL START BUTTON RELEASED
               00720 FOREVER
408A: AD 69 4B 00730          LDA GAME
408D: F0 0C    00740          BEQ ENDGAME
408F: AD 1F D0 00750          LDA CONSOL
4092: C9 06    00760          CMP #$06
4094: F0 DB    00770          BEQ START
4096: D0 F2    00780          BNE FOREVER
4098: 4C 8A 40 00790          JMP FOREVER
               00800 ;
               00810 ENDGAME
409B: A9 00    00820          LDA #$00
409D: A2 08    00830          LDX #$08
               00840 .1
409F: 9D 00 D2 00850          STA AUDF1,X ; TURN OFF ANY SOUNDS
40A2: CA       00860          DEX
40A3: 10 FA    00870          BPL .1
40A5: A2 13    00880          LDX #$13
```

```
                00890 .2
40A7: BD C9 4A  00900         LDA EMSG,X
40AA: 9D B8 71  00910         STA LINE3,X ; MOVE GAME OVER MESSAGE TO SCREEN
40AD: CA        00920         DEX
40AE: 10 F7     00930         BPL .2
40B0: A9 06     00940         LDA #$06
                00950 .3
40B2: CD 1F D0  00960         CMP CONSOL  ; START PRESSED?
40B5: D0 FB     00970         BNE .3      ; WAIT TILL IT IS
40B7: 4C 71 40  00980         JMP START
                00120         .IN "D:TANKVBI.SRC"
                00010 ;
                00020 * THIS IS D:TANKVBI.SRC
                00030 VBI
40BA: D8        00040         CLD         ; PRECAUTION
                00050 ; SET UP DLI VECTOR & ENABLE
40BB: A9 2B     00060         LDA #DLI
40BD: 8D 00 02  00070         STA VDSLST
40C0: A9 45     00080         LDA /DLI
40C2: 8D 01 02  00090         STA VDSLST+1
40C5: A9 C0     00100         LDA #$C0
40C7: 8D 0E D4  00110         STA NMIEN
40CA: AD 6A 4B  00120         LDA ENABLE  ; GAME/PAUSE FLAG
40CD: D0 03     00130         BNE PLAY    ; IF ON THEN OK
40CF: 4C 28 45  00140         JMP XVBI    ; ELSE GO TO EXIT
                00150 PLAY
40D2: A9 70     00160         LDA #$70
40D4: 85 4D     00170         STA ATRACT  ; DEFEAT ATTRACT MODE
40D6: A2 01     00180         LDX #$01
                00190 .1
40D8: BD 8E 4B  00200         LDA IMMUNE,X
40DB: F0 03     00210         BEQ .2
40DD: DE 8E 4B  00220         DEC IMMUNE,X ; COUNTDOWN IMMUNITY TIMER IF ON
                00230 .2
40E0: CA        00240         DEX
40E1: 10 F5     00250         BPL .1
                00260 HITCK
                00270 * COLLISION CHECK
40E3: A2 01     00280         LDX #$01
                00290 .1
40E5: 8E 66 4B  00300         STX SAVEX
40E8: BD 00 D0  00310         LDA MOPF,X  ; MISSILE HIT WALL?
40EB: F0 4D     00320         BEQ .3      ; SKIP IF NO COLLISION
                00330 ; GET POSITION
40ED: BC 87 4B  00340         LDY SHOTV,X ; GET SHOT VERTICAL POS.
40F0: B9 00 63  00350         LDA MISSILES,Y  ;GET MISSILE DATA FROM SCREEN
40F3: 3D 7B 4A  00360         AND SHTMSK,X    ; MASK MISSILE OFF
40F6: 99 00 63  00370         STA MISSILES,Y  ; ERASE SHOT
40F9: A9 00     00380         LDA #$00
40FB: 9D 89 4B  00390         STA SHOTF,X
40FE: 38        00400         SEC
40FF: BD 87 4B  00410         LDA SHOTV,X
4102: E9 20     00420         SBC #$20    ; -TOP
4104: 4A        00430         LSR         ; /2
4105: 4A        00440         LSR         ; /4
4106: 4A        00450         LSR         ; /8
4107: A2 14     00460         LDX #$14    ; 20 CHRS PER ROW
4109: 20 CC 46  00470         JSR MULTIPLY      ; GET OFSET FROM TOP OF SCREEN
410C: 18        00480         CLC         ; ADD TO SCREEN TO GET ADDRESS OF SCREEN ROW
410D: AD 59 4B  00490         LDA RESULT
4110: 69 00     00500         ADC #SCREEN
4112: 85 F7     00510         STA POTMP1
```

```
4114: AD 5A 4B 00520          LDA RESULT+1
4117: 69 70    00530          ADC /SCREEN
4119: 85 F8    00540          STA POTMP2
411B: AE 66 4B 00550          LDX SAVEX   ; GET BACK LOOP COUNTER
411E: 38       00560          SEC
411F: BD 85 4B 00570          LDA SHOTH,X
4122: E9 30    00580          SBC #$30    ; - LEFT EDGE OFFSET
4124: 4A       00590          LSR         ; /2
4125: 4A       00600          LSR         ; /4
4126: 4A       00610          LSR         ; /8 COLOR COLOR CLOCKS FOR CHR POS.
4127: A8       00620          TAY
4128: B1 F7    00630          LDA (POTMP1),Y
412A: 29 3F    00640          AND #$3F    ; MASK OFF COLOR
412C: C9 07    00650          CMP #$07    ; BORDER?
412E: 90 0A    00660          BCC .3      ; NO EFFECT
4130: AA       00670          TAX
4131: CA       00680          DEX         ; ASSUME A FRACTURE
4132: 4A       00690          LSR         ; EVEN OR ODD?
4133: B0 02    00700          BCS .2      ; OK
4135: A2 00    00710          LDX #$00    ; IF ALREADY FRACTURED THEN ERASE
               00720 .2
4137: 8A       00730          TXA
4138: 91 F7    00740          STA (POTMP1),Y
               00750 .3
413A: AE 66 4B 00760          LDX SAVEX
413D: CA       00770          DEX
413E: 10 A5    00780          BPL .1
               00790 ; CHECK IF TANK SHOT
4140: A2 01    00800          LDX #$01    ; TANK #
4142: A0 02    00810          LDY #$02  ; TANK # TIMES 2 TO INDEX A TABLE OF 2
4144: 8C 67 4B 00820          STY SAVEY                       BYTE ADDRESSES
               00830 .4
4147: BD 8B 4B 00840          LDA CYCLES,X ; TANK ALREADY HIT?
414A: 1D 8E 4B 00850          ORA IMMUNE,X ; OR HAVE TEMPORARY IMMUNITY?
414D: F0 03    00860          BEQ .42
414F: 4C D7 41 00870          JMP .7      ; DON'T BOTHER
               00880 .42
4152: BC 7F 4A 00890          LDY WHO,X   ; GET OPPONENTS TANK #
4155: B9 08 D0 00900          LDA MOPL,Y  ; SHOT?
4158: 3D 7D 4A 00910          AND HITMASK,X  ; MASK OFF SHOOTING SELF
415B: D0 11    00920          BNE .41
415D: B9 8B 4B 00930          LDA CYCLES,Y ; OTHER PLAYER BLOWING UP
4160: F0 04    00940          BEQ .40     ; NO.
4162: C9 FF    00950          CMP #$FF    ; FIRST CYCLE?
4164: D0 71    00960          BNE .7      ; NOT FIRST CYCLE, IGNORE DEBRIS COLLISION
               00970 .40
4166: BD 0E D0 00980          LDA P2PL,X  ; CRASHED?
4169: 39 7D 4A 00990          AND HITMASK,Y ; MASK OFF SHOOTING SELF
416C: F0 69    01000          BEQ .7      ; NO HIT
               01010 ; FIND UP ZAP ARRAY AND DOWN ZAP ARRAY
               01015 ; AND FILL WITH CURRENT VERTICAL POSITION
               01020 .41
416E: F8       01030          SED
416F: 18       01040          CLC
4170: B9 93 4B 01050          LDA SCORE,Y ; UPDATE SCORE IN DECIMAL
4173: 69 01    01060          ADC #$01
4175: 99 93 4B 01070          STA SCORE,Y
4178: D8       01080          CLD
               01090 ; THIS SECTION INTIALIZES THE TABLES FOR DZAP AND UZAP
               01095 ; TO THE TANK'S CURRENT VERTICAL POSITION
4179: AC 67 4B 01100          LDY SAVEY
417C: B9 A5 4A 01110          LDA UZTAB,Y
```

```
417F: 85 F7    01120          STA POTMP1
4181: B9 A6 4A 01130          LDA UZTAB+1,Y
4184: 85 F8    01140          STA POTMP2
4186: B9 A9 4A 01150          LDA DZTAB,Y
4189: 85 F9    01160          STA POTMP3
418B: B9 AA 4A 01170          LDA DZTAB+1,Y
418E: 85 FA    01180          STA POTMP4
4190: A0 08    01190          LDY #$08
4192: BD 6D 4B 01200          LDA PLAYERV,X   ; TANK VERTICAL POS.
               01210 .5
4195: 91 F7    01220          STA (POTMP1),Y  ; UZAP TABLE
4197: 91 F9    01230          STA (POTMP3),Y  ; DZAP TABLE
4199: 88       01240          DEY
419A: D0 F9    01250          BNE .5
               01260 ; IF 1ST PLAYER HIT ERASE 2ND PLAYERS SHOT & VICE VERSA
419C: BC 7F 4A 01270          LDY WHO,X
419F: A9 00    01280          LDA #$00
41A1: 99 89 4B 01290          STA SHOTF,Y
41A4: 99 85 4B 01300          STA SHOTH,Y
41A7: 99 04 D0 01310          STA HPOSMO,Y
41AA: B9 87 4B 01320          LDA SHOTV,Y
41AD: A8       01330          TAY
41AE: A9 00    01340          LDA #$00
41B0: 99 00 63 01350          STA MISSILES,Y
               01360 ; ERASE HIT TANK
41B3: BD 6D 4B 01370          LDA PLAYERV,X
41B6: 85 F7    01380          STA POTMP1
41B8: 85 F9    01390          STA POTMP3
41BA: BD 75 4A 01400          LDA PLAYTAB,X
41BD: 85 F8    01410          STA POTMP2
41BF: BD 77 4A 01420          LDA PLAYTAB2,X
41C2: 85 FA    01430          STA POTMP4
41C4: A9 00    01440          LDA #$00
41C6: A0 0F    01450          LDY #$0F
               01460 .6
41C8: 91 F7    01470          STA (POTMP1),Y
41CA: 91 F9    01480          STA (POTMP3),Y
41CC: 88       01490          DEY
41CD: 10 F9    01500          BPL .6
41CF: AC 67 4B 01510          LDY SAVEY
41D2: A9 FF    01520          LDA #$FF
41D4: 9D 8B 4B 01530          STA CYCLES,X
               01540 .7
41D7: CE 67 4B 01550          DEC SAVEY
41DA: CE 67 4B 01560          DEC SAVEY
41DD: CA       01570          DEX
41DE: 30 03    01580          BMI XHITCK
41E0: 4C 47 41 01590          JMP .4      ; BRANCH OUT OF RANGE, JUMP USED
               01600 XHITCK
41E3: 8D 1E D0 01610          STA HITCLR  ; CLEAR COLLISIONS
               01620 * UPDATE ANY TANK EXPLOSION
               01630 EXPLODE
41E6: A0 02    01640          LDY #$02
41E8: 8C 67 4B 01650          STY SAVEY
41EB: A2 01    01660          LDX #$01
41ED: 8E 66 4B 01670          STX SAVEX
               01680 .1
41F0: BD 8B 4B 01690          LDA CYCLES,X
41F3: D0 03    01700          BNE .10
41F5: 4C 79 42 01710          JMP .5      ; BRANCH OUT OF RANGE, JUMP USED
               01720 .10
41F8: DE 8B 4B 01730          DEC CYCLES,X
```

```
41FB: D0 13    01740          BNE .11     ; IF NOT LAST TIME THROUGH EXPLOSION CYCLE
41FD: BD B1 4A 01750          LDA STARTH,X      ; RESET TANK POS.
4200: 9D 6B 4B 01760          STA PLAYERH,X
4203: BD B3 4A 01770          LDA STARTV,X
4206: 9D 6D 4B 01780          STA PLAYERV,X
4209: A9 FF    01790          LDA #$FF    ; SET IMMUNITY TIMER TO APROX. 4 SEC.
420B: 9D 8E 4B 01800          STA IMMUNE,X
420E: D0 69    01810          BNE .5      ; ALWAYS.
               01820 .11
4210: AD 0A D2 01830          LDA RANDOM  ; PICK A RANDOM COLOR
4213: 9D 14 D0 01840          STA COLPM2,X   ; USE TO FLASH TANK COLOR (EXPLOSION COLOR)
               01850 * GET ADDRESSES OF UZAP AND DZAP TABLES AND PLACE ON PAGE
4216: B9 A5 4A 01860          LDA UZTAB,Y                          0 FOR ACCESS
4219: 85 F7    01870          STA POTMP1
421B: B9 A6 4A 01880          LDA UZTAB+1,Y
421E: 85 F8    01890          STA POTMP2
4220: B9 A9 4A 01900          LDA DZTAB,Y
4223: 85 F9    01910          STA POTMP3
4225: B9 AA 4A 01920          LDA DZTAB+1,Y
4228: 85 FA    01930          STA POTMP4
               01940 * POTMP1&2 HAVE ADDRESS OF UZAP ARRAY,POTMP3&4 HAVE ADDRESS
               01950 * SET POTMP5&6 TO ADDRESS OF PLAYER          OF DZAP ARRAY.
422A: A9 00    01960          LDA #$00
422C: 85 FB    01970          STA POTMP5
422E: BD 75 4A 01980          LDA PLAYTAB,X
4231: 85 FC    01990          STA POTMP6
4233: A9 08    02000          LDA #$08
4235: 8D 8D 4B 02010          STA INDEX
               02020 .2
4238: AC 8D 4B 02030          LDY INDEX
423B: B1 F7    02040          LDA (POTMP1),Y    ; GET VERTICAL POS. OF OLD LINE IN UZAP
423D: A8       02050          TAY               ; USE TO INDEX INTO PLAYER RAM
423E: A9 00    02060          LDA #$00
4240: 91 FB    02070          STA (POTMP5),Y    ; ERASE OLD LINE
4242: AC 8D 4B 02080          LDY INDEX
4245: B1 F9    02090          LDA (POTMP3),Y    ; GET VERTICAL POS. OF OLD LINE IN DZAP
4247: A8       02100          TAY               ; FIND OLD BYTE IN PLAYER RAM
4248: A9 00    02110          LDA #$00
424A: 91 FB    02120          STA (POTMP5),Y    ; ERASE OLD LINE
424C: AC 8D 4B 02130          LDY INDEX
424F: 38       02140          SEC
4250: B1 F7    02150          LDA (POTMP1),Y   ; GET V.POS. OF ELEMENT IN UZAP
4252: ED 8D 4B 02160          SBC INDEX        ; CALCULATE NEW POS.
4255: C9 20    02170          CMP #$20         ; TOO HIGH?
4257: 90 07    02180          BCC .3
4259: 91 F7    02190          STA (POTMP1),Y   ; SAVE BACK IN UZAP ARRAY
425B: A8       02200          TAY              ; USE TO INDEX INTO PLAYER RAM
425C: A9 FF    02210          LDA #$FF         ; VALUE OF A LINE
425E: 91 FB    02220          STA (POTMP5),Y   ; PUT NEW LINE ON
               02230 .3
               02240 * DOWN WORKS THE SAME AS UP EXCEPT THAT THE INDEX IS ADDED TO THE
               02245 * ARRAY INSTEAD OF SUBTRACTED
4260: AC 8D 4B 02250          LDY INDEX
4263: 18       02260          CLC
4264: B1 F9    02270          LDA (POTMP3),Y
4266: 6D 8D 4B 02280          ADC INDEX
4269: C9 C0    02290          CMP #$C0          ; TOO LOW?
426B: B0 07    02300          BCS .4
426D: 91 F9    02310          STA (POTMP3),Y
426F: A8       02320          TAY
4270: A9 FF    02330          LDA #$FF
4272: 91 FB    02340          STA (POTMP5),Y    ; DRAW NEW LINE
```

```
                02350 .4
4274: CE 8D 4B  02360         DEC INDEX
4277: D0 BF     02370         BNE .2
                02380 .5
4279: AC 67 4B  02390         LDY SAVEY
427C: 88        02400         DEY
427D: 88        02410         DEY
427E: CA        02420         DEX
427F: 30 03     02430         BMI MOVSHELL
4281: 4C F0 41  02440         JMP .1      ; BRANCH OUT OF RANGE, JMP USED
                02450 * MOVE MISSILES
                02460 MOVSHELL
4284: A2 01     02470         LDX #$01
                02480 .1
4286: BD 89 4B  02490         LDA SHOTF,X       ; SHOT IN PROGRESS?
4289: F0 30     02500         BEQ .3            ; NO UPDATE
428B: BC 87 4B  02510         LDY SHOTV,X
428E: B9 00 63  02520         LDA MISSILES,Y    ; GET OLD MISSILE BYTE
4291: 3D 7B 4A  02530         AND SHTMSK,X      ; MASK OFF SHOT KEEP ANY OTHER SHOT ON
4294: 99 00 63  02540         STA MISSILES,Y    ; SAVE IT BACK
4297: BC 83 4B  02550         LDY SHOTD,X       ; GET SHOT DIRECTION
429A: 18        02560         CLC
429B: BD 85 4B  02570         LDA SHOTH,X       ; GET SHOT HORIZ. POS.
429E: 79 5C 4A  02580         ADC HOFFS,Y       ; CALCULATE NEW HORZ. POS.
                02590 * THE LINES HERE THAT ARE REMed OUT ARE NOT NEEDED AS LONG
                02595 * AS THERE IS A SCREEN BORDER FOR THE SHOT TO HIT
                02600 *******  CMP #$30          ; LEFT EDGE
                02610 *******  BCC .2
                02620 *******  CMP #$C8          ; RIGHT EDGE
                02630 ******** BCS .2
42A1: 9D 85 4B  02640         STA SHOTH,X       ; SAVE NEW SHOT HORIZ. POS.
42A4: 9D 04 D0  02650         STA HPOSM0,X      ; TELL ANTIC NEW H. POS.
42A7: 18        02660         CLC
42A8: BD 87 4B  02670         LDA SHOTV,X       ; CALCULATE NEW VERT. POS.
42AB: 79 64 4A  02680         ADC VOFFS,Y
                02690 ******** CMP #$20
                02700 ******** BCC .2
                02710 ******** CMP #$B0
                02720 ******** BCS .2
42AE: 9D 87 4B  02730         STA SHOTV,X       ; SAVE IT BACK
42B1: A8        02740         TAY               ; USE TO INDEX INTO MISSILE RAM
42B2: BD 79 4A  02750         LDA SHELLS,X      ; DATA FOR SHOT IMAGE
42B5: 19 00 63  02760         ORA MISSILES,Y    ; MERGE WITH ANY OTHER MISSILE DATA
42B8: 99 00 63  02770         STA MISSILES,Y    ; PUT ON SCREEN
                02780 *******  BNE .3           ; ALWAYS
                02790 .2
                02800 ******** LDA #$00
                02810 ******** STA HPOSM0,X
                02820 ******** STA SHOTF,X
                02830 .3
42BB: CA        02840         DEX
42BC: 10 C8     02850         BPL .1
                02860 *
                02870 *
                02880 *
                02890 UPDATE
42BE: A5 14     02900         LDA RTCLOC
42C0: 29 02     02910         AND #$02
42C2: F0 03     02920         BEQ .1
42C4: 4C 55 44  02930         JMP XTANK
                02940 .1
42C7: A5 14     02950         LDA RTCLOC
```

```
42C9: 29 01    02960          AND #$01
42CB: 8D 68 4B 02970          STA TANK          ; EACH TANK IS HANDLED EVERY OTHER VBI
42CE: AA       02980          TAX
42CF: BD 8B 4B 02990          LDA CYCLES,X      ; DEAD??
42D2: F0 03    03000          BEQ .2
42D4: 4C 55 44 03010          JMP XTANK
                03020 .2
42D7: BD 7D 4B 03030          LDA TURRETF,X     ; CHECK TURRET FLAG
42DA: D0 08    03040          BNE DOTURRET      ; IF FLAG ON CONTINUE IN TURRET ROUTINE
42DC: BD 84 02 03050          LDA STRIGO,X      ; CHECK IF BUTTON DOWN
42DF: F0 03    03060          BEQ DOTURRET      ; BUTTON DOWN
42E1: 4C 6E 43 03070          JMP DOTANK
                03080 DOTURRET
42E4: BD 84 02 03090          LDA STRIGO,X      ; IS BUTTON STILL HELD DOWN
42E7: F0 41    03100          BEQ ROTATE        ; THEN KEEP ROTATING TURRET
                03110 * OTHERWISE BUTTON RELEASED-SHOOT
42E9: A9 00    03120          LDA #$00
42EB: 9D 7D 4B 03130          STA TURRETF,X     ; RETURN TO NORMAL JOYSTICK MOVEMENT
42EE: BD 89 4B 03140          LDA SHOTF,X       ; SHOT IN PROGRESS?
42F1: D0 7B    03150          BNE DOTANK        ; NO SHOT GO TO TANK MODE
42F3: BD 7F 4B 03160          LDA TURRETD,X
42F6: 9D 83 4B 03170          STA SHOTD,X       ; SHOT DIRECTION=TURRET DIRECTION
42F9: C9 05    03180          CMP #$05    ; SHOT COMING OUT RIGHT SIDE OF TANK?
                03190 * SET SHOT POS. TO CENTER OF TANK
42FB: BD 6B 4B 03200          LDA PLAYERH,X
42FE: B0 03    03210          BCS .1      ; IF SHOT OUT OF LEFT SIDE OF TANK
4300: 38       03220          SEC
4301: E9 02    03230          SBC #$02    ; START SHOT OVER TO THE LEFT
                03240 .1
4303: 18       03250          CLC
4304: 69 04    03260          ADC #$04
4306: 9D 85 4B 03270          STA SHOTH,X
4309: 9D 04 D0 03280          STA HPOSM0,X
430C: 18       03290          CLC
430D: BD 6D 4B 03300          LDA PLAYERV,X
4310: 69 08    03310          ADC #$08
4312: 9D 87 4B 03320          STA SHOTV,X
4315: A8       03330          TAY
4316: BD 79 4A 03340          LDA SHELLS,X
4319: 19 00 63 03350          ORA MISSILES,Y
431C: 99 00 63 03360          STA MISSILES,Y
431F: 9D 89 4B 03370          STA SHOTF,X       ; SET SHOT FLAG TO NON-ZERO
                03380 * SET FLAG FOR SHOT SOUND
4322: A9 10    03390          LDA #$10
4324: 9D 81 4B 03400          STA SHOTS,X
4327: 4C 55 44 03410          JMP XTANK   ; LEAVE
                03420 ; DRAW TURRET POSITION THIS FRAME
                03430 ROTATE
432A: A9 01    03440          LDA #$01
432C: 9D 7D 4B 03450          STA TURRETF,X     ; SET TURRET MODE ON
432F: A5 14    03460          LDA RTCLOC        ;TURN TURRET EVERY 8TH JIFFY
4331: 29 04    03470          AND #$04
4333: D0 0F    03480          BNE .1
4335: 18       03490          CLC
4336: BC 78 02 03500          LDY STICK0,X
4339: B9 4C 4A 03510          LDA DOFFS,Y
433C: 7D 7F 4B 03520          ADC TURRETD,X
433F: 29 07    03530          AND #$07    ; KEEP DIR 0-7
4341: 9D 7F 4B 03540          STA TURRETD,X
4344: BD 6B 4B 03550 .1       LDA PLAYERH,X
4347: 8D 73 4B 03560          STA NEWH    ; KEEP TEMP VALUES BECAUSE WE CAN'T CHANGE REAL
434A: BD 6D 4B 03570          LDA PLAYERV,X ;              -VALUES BACK IF NEW POS. INVALID
```

```
434D: 8D 74 4B 03580          STA NEWV    ;
4350: BD 7F 4B 03590          LDA TURRETD,X
4353: 8D 77 4B 03600          STA NEWTD
4356: BD 71 4B 03610          LDA PLAYERF,X ;GET CURRENT DIRECTION
4359: 8D 76 4B 03620          STA NEWF    ;TEMP VALUE
435C: 8D 75 4B 03630          STA NEWD
435F: 0A       03640          ASL         ; INDEX INTO TANK SHAPES TABLE
4360: A8       03650          TAY
4361: B9 8A 49 03660          LDA TANKTAB,Y
4364: 85 F7    03670          STA POTMP1
4366: B9 8B 49 03680          LDA TANKTAB+1,Y
4369: 85 F8    03690          STA POTMP1+1  ; WE NOW HAVE A PAGE 0 POINTER TO NEW
436B: 4C B8 43 03700          JMP MID                                    \TANK IMAGE
               03710 ;
               03720 ;
               03730 DOTANK
436E: AE 68 4B 03740          LDX TANK    ; INDEX TO WHICH TANK
4371: BC 78 02 03750          LDY STICKO,X      ; READ STICK
4374: BD 6B 4B 03760          LDA PLAYERH,X
4377: 8D 73 4B 03770          STA NEWH    ; KEEP TEMP VALUES BECAUSE WE CAN'T CHANGE-
437A: BD 6D 4B 03780          LDA PLAYERV,X ;   -REALVALUES BACK IF NEW POS. INVALID
437D: 8D 74 4B 03790          STA NEWV    ;
4380: BD 7F 4B 03800          LDA TURRETD,X
4383: 8D 77 4B 03810          STA NEWTD
4386: A5 14    03820          LDA RTCLOC
4388: 29 1C    03830          AND #$1C    ; SLOW DOWN ROTATION OF TANK
438A: 08       03840          PHP         ; SAVE PROCESSOR STATUS
438B: BD 71 4B 03850          LDA PLAYERF,X   ; GET CURRENT TANK FACING
438E: 28       03860          PLP         ; GET BACK DELAY TIMER STATUS
438F: D0 15    03870          BNE .0   ; LEAVE IF TO SOON TO ROTATE WITH OLD FACING
4391: 18       03880          CLC                                    VALUE IN ACCUM.
4392: BD 7F 4B 03890          LDA TURRETD,X ; TURRET ROTATES WITH BODY OF TANK
4395: 79 4C 4A 03900          ADC DOFFS,Y       ; +1,0, OR -1
4398: 29 07    03910          AND #$07          ; KEEP TURRET DIRECTION 0-7
439A: 8D 77 4B 03920          STA NEWTD
439D: 18       03930          CLC
439E: BD 71 4B 03940          LDA PLAYERF,X     ; GET CURRENT DIRECTION
43A1: 79 4C 4A 03950          ADC DOFFS,Y       ; ROTATE LEFT OR RIGHT DEPENDING ON STICK
43A4: 29 07    03960          AND #$07          ; KEEP DIR 0-7
               03970 .0
43A6: 8D 76 4B 03980          STA NEWF          ; TEMP VALUE
43A9: 8D 75 4B 03990          STA NEWD
43AC: 0A       04000          ASL         ; INDEX INTO TANK SHAPES TABLE
43AD: A8       04010          TAY
43AE: B9 8A 49 04020          LDA TANKTAB,Y
43B1: 85 F7    04030          STA POTMP1
43B3: B9 8B 49 04040          LDA TANKTAB+1,Y
43B6: 85 F8    04050          STA POTMP1+1  ; WE NOW HAVE A PAGE 0 POINTER TO NEW TANK IMAGE
43B8: BD 78 02 04060 MID      LDA STICKO,X
43BB: 29 03    04070          AND #$03    ; KEEP U/D BITS
43BD: C9 03    04080          CMP #$03    ; NEUTRAL
43BF: F0 36    04090          BEQ DRTANK  ; DRAW TANK THEN
43C1: C9 01    04100          CMP #$01    ; DOWN BIT OFF?
43C3: D0 0B    04110          BNE .1      ; NO.
43C5: 18       04120          CLC
43C6: AD 76 4B 04130          LDA NEWF
43C9: 69 04    04140          ADC #$04    ; DIRECTION IS 180 DEG. OPPOSITE FACING
43CB: 29 07    04150          AND #$07    ; KEEP DIR 0-7
43CD: 8D 75 4B 04160          STA NEWD
               04170 .1
43D0: AC 75 4B 04180          LDY NEWD
43D3: 18       04190          CLC
```

```
43D4: BD 6B 4B 04200          LDA PLAYERH,X
43D7: 79 5C 4A 04210          ADC HOFFS,Y
43DA: C9 C8    04220          CMP #$C8    ; RIGHT EDGE
43DC: B0 77    04230          BCS XTANK
43DE: C9 30    04240          CMP #$30    ; LEFT EDGE
43E0: 90 73    04250          BCC XTANK
43E2: 8D 73 4B 04260          STA NEWH
43E5: 18       04270          CLC
43E6: BD 6D 4B 04280          LDA PLAYERV,X
43E9: 79 64 4A 04290          ADC VOFFS,Y
43EC: C9 B0    04300          CMP #$B0    ; BOTTOM
43EE: B0 65    04310          BCS XTANK
43F0: C9 20    04320          CMP #$20    ; TOP
43F2: 90 61    04330          BCC XTANK
43F4: 8D 74 4B 04340          STA NEWV
               04350 DRTANK
43F7: 20 10 46 04360          JSR LOOKAHEAD     ; CHECK FOR LEGAL MOVE
43FA: B0 2E    04370          BCS DRTURRET      ; POSITION NO GOOD
43FC: AE 68 4B 04380          LDX TANK
43FF: AD 77 4B 04390          LDA NEWTD
4402: 9D 7F 4B 04400          STA TURRETD,X
4405: AD 76 4B 04410          LDA NEWF
4408: 9D 71 4B 04420          STA PLAYERF,X
440B: AD 73 4B 04430          LDA NEWH
440E: 9D 6B 4B 04440          STA PLAYERH,X
4411: 9D 00 D0 04450          STA HPOSP0,X      ; TELL ANTIC NEW POS.
4414: AD 74 4B 04460          LDA NEWV
4417: 9D 6D 4B 04470          STA PLAYERV,X
441A: 85 F9    04480          STA POTMP3        ; BUILD PLAYER ADDRESS ON PAGE0
441C: BD 75 4A 04490          LDA PLAYTAB,X
441F: 85 FA    04500          STA POTMP4
4421: A0 0F    04510          LDY #$0F          ; 16 ELEMENTS
               04520 .2
4423: B1 F7    04530          LDA (POTMP1),Y    ; GET A BYTE OF TANK DATA
4425: 91 F9    04540          STA (POTMP3),Y    ; PUT IN PLAYER RAM
4427: 88       04550          DEY
4428: 10 F9    04560          BPL .2            ; TILL ALL BYTES ARE MOVED
               04570 ; TANK DRAWN, NOW DO TURRET
               04580 DRTURRET
442A: AE 68 4B 04590          LDX TANK
442D: BD 7F 4B 04600          LDA TURRETD,X
4430: 0A       04610          ASL
4431: A8       04620          TAY
4432: B9 1A 4A 04630          LDA TURTAB,Y
4435: 85 F7    04640          STA POTMP1
4437: B9 1B 4A 04650          LDA TURTAB+1,Y
443A: 85 F8    04660          STA POTMP2
443C: BD 6D 4B 04670          LDA PLAYERV,X
443F: 85 F9    04680          STA POTMP3
4441: BD 77 4A 04690          LDA PLAYTAB2,X
4444: 85 FA    04700          STA POTMP4
4446: A0 0F    04710          LDY #$0F
               04720 .1
4448: B1 F7    04730          LDA (POTMP1),Y    ; GET TURRET DATA
444A: 91 F9    04740          STA (POTMP3),Y    ; STORE IN PLAYER RAM
444C: 88       04750          DEY
444D: 10 F9    04760          BPL .1
444F: BD 6B 4B 04770          LDA PLAYERH,X     ; GET HORIZ. POS.
4452: 9D 02 D0 04780          STA HPOSP2,X      ; TELL ANTIC
               04790 ;
               04800 XTANK
               04810 ;
```

```
                04820 SOUND
4455: A0 02     04830           LDY #$02
4457: A2 01     04840           LDX #$01
                04850 .1
4459: BD 81 4B  04860           LDA SHOTS,X
445C: F0 0E     04870           BEQ .2
445E: DE 81 4B  04880           DEC SHOTS,X
4461: BD 81 4B  04890           LDA SHOTS,X
4464: 99 05 D2  04900           STA AUDC3,Y
4467: A9 20     04910           LDA #$20
4469: 99 04 D2  04920           STA AUDF3,Y
                04930 .2
446C: 88        04940           DEY
446D: 88        04950           DEY
446E: CA        04960           DEX
446F: 10 E8     04970           BPL .1
                04980 SOUND2
4471: A2 01     04990           LDX #$01
4473: A0 02     05000           LDY #$02
                05010 .1
4475: BD 8B 4B  05020           LDA CYCLES,X      ; TANK DYING
4478: F0 15     05030           BEQ .2       ; DO MOTOR SOUND IF NOT
447A: 49 FF     05040           EOR #$FF
447C: 99 00 D2  05050           STA AUDF1,Y
447F: 49 FF     05060           EOR #$FF
4481: 4A        05070           LSR
4482: 4A        05080           LSR
4483: 4A        05090           LSR
4484: 4A        05100           LSR
4485: 4A        05110           LSR
4486: 29 0F     05120           AND #$0F     ; KEEP VOLUME IN RANGE
4488: 09 20     05130           ORA #$20     ; SET DISTORTION
448A: 99 01 D2  05140           STA AUDC1,Y
448D: D0 1A     05150           BNE .3       ; ALWAYS. SKIP MOTOR SOUND
                05160 .2
448F: A5 21     05170           LDA $21      ; SET DISTORTION, LOW VOLUME
4491: 99 01 D2  05180           STA AUDC1,Y
4494: BD AD 4A  05190           LDA IDLE,X   ; ASSUME IDLE TONE
4497: 99 00 D2  05200           STA AUDF1,Y
449A: BD 78 02  05210           LDA STICK0,X
449D: 29 03     05220           AND #$03     ; KEEP UP/DOWN BITS
449F: C9 03     05230           CMP #$03     ; MOVING?
44A1: F0 06     05240           BEQ .3       ; NO
44A3: BD AF 4A  05250           LDA VROOM,X ; ENCREASE PITCH
44A6: 99 00 D2  05260           STA AUDF1,Y
                05270 .3
44A9: 88        05280           DEY
44AA: 88        05290           DEY
44AB: CA        05300           DEX
44AC: 10 C7     05310           BPL .1
                05320 ;
                05330 COUNTDOWN
44AE: F8        05340           SED
44AF: AD 91 4B  05350           LDA MINUTES
44B2: 0D 92 4B  05360           ORA SECONDS
44B5: F0 50     05370           BEQ XCOUNT   ; LEAVE IF 0
44B7: CE 90 4B  05380           DEC TIMER
44BA: D0 4B     05390           BNE XCOUNT
44BC: A9 3C     05400           LDA #60
44BE: 8D 90 4B  05410           STA TIMER    ; 60 VBLANKS PER SEC.
44C1: 38        05420           SEC
44C2: AD 92 4B  05430           LDA SECONDS
```

```
44C5: E9 01    05440          SBC #$01
44C7: 8D 92 4B 05450          STA SECONDS
44CA: D0 1D    05460          BNE .2
44CC: A9 59    05470          LDA #$59
44CE: 8D 92 4B 05480          STA SECONDS ; RESET # OF SECONDS TO 59 DEC.
44D1: AD 91 4B 05490          LDA MINUTES ; CHECK IF MINUTES ALREADY AT 0
44D4: D0 0A    05500          BNE .1      ; OK
44D6: A9 00    05510          LDA #$00
44D8: 8D 69 4B 05520          STA GAME    ; FLAG GAME NOT IN PROGRESS
44DB: 8D 6A 4B 05530          STA ENABLE  ; SET VBI FLAG TO SKIP GAME ROUTINES
44DE: F0 09    05540          BEQ .2      ; ALWAYS
               05550 .1
44E0: 38       05560          SEC
44E1: AD 91 4B 05570          LDA MINUTES
44E4: E9 01    05580          SBC #$01
44E6: 8D 91 4B 05590          STA MINUTES
               05600 .2
44E9: AD 91 4B 05610          LDA MINUTES
44EC: 09 10    05620          ORA #$10    ; TURN DECIMAL # INTO A CHARACTER
44EE: 8D C0 71 05630          STA LINE3+8
44F1: AD 92 4B 05640          LDA SECONDS
44F4: 4A       05650          LSR         ; GET 10'S
44F5: 4A       05660          LSR
44F6: 4A       05670          LSR
44F7: 4A       05680          LSR
44F8: 09 10    05690          ORA #$10    ; MAKE A CHR
44FA: 8D C2 71 05700          STA LINE3+10
44FD: AD 92 4B 05710          LDA SECONDS
4500: 29 0F    05720          AND #$0F    ; GET 1'S
4502: 09 10    05730          ORA #$10
4504: 8D C3 71 05740          STA LINE3+11
               05750 XCOUNT
4507: D8       05760          CLD
4508: A2 01    05770          LDX #$01
               05780 .1
450A: BC 05 4B 05790          LDY SCPOS,X
450D: BD 93 4B 05800          LDA SCORE,X       ; PRINT SCORE ON SCREEN
4510: 4A       05810          LSR
4511: 4A       05820          LSR
4512: 4A       05830          LSR
4513: 4A       05840          LSR
4514: 1D 57 4B 05850          ORA CSHIFT,X      ; MAKE CHR+COLOR#
4517: 99 A4 71 05860          STA LINE2,Y
451A: BD 93 4B 05870          LDA SCORE,X
451D: 29 0F    05880          AND #$0F
451F: 1D 57 4B 05890          ORA CSHIFT,X
4522: 99 A5 71 05900          STA LINE2+1,Y
4525: CA       05910          DEX
4526: 10 E2    05920          BPL .1
               05930 ;
               05940 XVBI
4528: 4C 62 E4 05950          JMP XITVBV
               05960 ;
               00130          .IN "D:TANKSUBS.SRC"
               00010 * MISC TANK SUBROUTINES
               00020 * THIS IS D:TANKSUBS.SRC
               00030 ;
               00040 DLI
452B: 48       00050          PHA
452C: A9 E0    00060          LDA /ATARI
452E: 8D 0A D4 00070          STA WSYNC
4531: 8D 09 D4 00080          STA CHBASE
```

```
4534: 68       00090          PLA
4535: 40       00100          RTI
               00110 ;
               00120 RESTART
4536: 20 B7 45 00130          JSR DOSCREEN      ; RESTORE SCREEN MAZE
               00140 *
               00150 * CLEAR GAME VARIABLES
4539: A9 00    00160          LDA #$00
453B: AA       00170          TAX
               00180 .0
453C: 9D 59 4B 00190          STA FIRSTVAR,X
453F: E8       00200          INX
4540: E0 3C    00210          CPX #LASTVAR-FIRSTVAR
4542: 90 F8    00220          BCC .0
4544: F0 F6    00230          BEQ .0
               00240 *
               00250 * REPOSITION TANKS
               00260 * FIRST ERASE ANY OLD DATA
4546: A9 00    00270          LDA #$00
4548: AA       00280          TAX
               00290 .1
4549: 9D 00 63 00300          STA MISSILES,X
454C: 9D 00 64 00310          STA PLAYER0,X
454F: 9D 00 65 00320          STA PLAYER1,X
4552: 9D 00 66 00330          STA PLAYER2,X
4555: 9D 00 67 00340          STA PLAYER3,X
4558: CA       00350          DEX
4559: D0 EE    00360          BNE .1
455B: AD B3 4A 00370          LDA STARTV  ; MIDSCREEN
455E: 8D 6D 4B 00380          STA PLAYERV
4561: 85 F7    00390          STA POTMP1
4563: AD B4 4A 00400          LDA STARTV+1
4566: 8D 6E 4B 00410          STA PLAYERV+1
4569: 85 F9    00420          STA POTMP3
456B: A9 64    00430          LDA /PLAYER0
456D: 85 F8    00440          STA POTMP2
456F: A9 65    00450          LDA /PLAYER1
4571: 85 FA    00460          STA POTMP4
4573: A0 0F    00470          LDY #$0F
               00480 .2
4575: B9 0A 49 00490          LDA TANK0,Y
4578: 91 F7    00500          STA (POTMP1),Y
457A: 91 F9    00510          STA (POTMP3),Y
457C: 88       00520          DEY
457D: 10 F6    00530          BPL .2
457F: AD B1 4A 00540          LDA STARTH
4582: 8D 6B 4B 00550          STA PLAYERH
4585: 8D 00 D0 00560          STA HPOSP0
4588: AD B2 4A 00570          LDA STARTH+1
458B: 8D 6C 4B 00580          STA PLAYERH+1
458E: 8D 01 D0 00590          STA HPOSP1
               00600 ;
               00610 * SET FLAGS FOR GAME IN PROGRESS
4591: A2 13    00620          LDX #$13
               00630 .3
4593: BD B5 4A 00640          LDA SMSG,X
4596: 9D B8 71 00650          STA LINE3,X
4599: CA       00660          DEX
459A: 10 F7    00670          BPL .3
459C: A9 60    00680          LDA #$60
459E: 8D 90 4B 00690          STA TIMER
45A1: A9 60    00700          LDA #$60    ; SET GAME TIME TO 3:00
```

```
45A3: 8D 92 4B 00710          STA SECONDS
45A6: A9 02    00720          LDA #$02
45A8: 8D 91 4B 00730          STA MINUTES
45AB: A9 01    00740          LDA #$01
45AD: 8D 1E D0 00750          STA HITCLR  ; CLEAR ANY ERRONEOUS COLLISIONS
45B0: 8D 69 4B 00760          STA GAME
45B3: 8D 6A 4B 00770          STA ENABLE
45B6: 60       00780          RTS
               00790 *
               00800 DOSCREEN
               00810 ; RESET SCREEN
45B7: A9 7A    00820          LDA #MAZE
45B9: 85 F6    00830          STA POTMP0
45BB: A9 47    00840          LDA /MAZE
45BD: 85 F7    00850          STA POTMP1
45BF: A9 00    00860          LDA #SCREEN
45C1: 85 F8    00870          STA POTMP2
45C3: A9 70    00880          LDA /SCREEN
45C5: 85 F9    00890          STA POTMP3
45C7: 18       00900          CLC
45C8: A9 90    00910          LDA #400    ; SET UP TO MOVE 400 BYTES TO SCREEN
45CA: 69 00    00920          ADC #SCREEN
45CC: 8D 78 4B 00930          STA TEMP0
45CF: A9 01    00940          LDA /400
45D1: 69 70    00950          ADC /SCREEN
45D3: 8D 79 4B 00960          STA TEMP1
45D6: A0 00    00970          LDY #$00
               00980 .2
45D8: B1 F6    00990          LDA (POTMP0),Y ; TAKE A BYTE
45DA: 91 F8    01000          STA (POTMP2),Y ; MOVE TO SCREEN MEM
45DC: E6 F6    01010          INC POTMP0     ; INCREMENT LOAD ADDRESS BY 1
45DE: D0 02    01020          BNE .3
45E0: E6 F7    01030          INC POTMP1     ; INCREMENT HIBYTE IF NECC.
               01040 .3
45E2: E6 F8    01050          INC POTMP2     ; INCREMENT STORE ADDRESS BY ONE
45E4: D0 02    01060          BNE .4
45E6: E6 F9    01070          INC POTMP3     ; INCREMENT HIBYTE IF NECCESSARY
               01080 .4
45E8: A5 F8    01090          LDA POTMP2
45EA: CD 78 4B 01100          CMP TEMP0      ; IS LOW BYTE=LOWBYTE OF ENDING ADDRESS
45ED: D0 E9    01110          BNE .2         ; NO? KEEP MOVING THEM
45EF: A5 F9    01120          LDA POTMP3
45F1: CD 79 4B 01130          CMP TEMP1      ; HIBYTE=HIBYTE OF ENDING ADDRESS?
45F4: D0 E2    01140          BNE .2         ; KEEP MOVING THEM TILL ENDING ADDRESS REACHED
45F6: A2 13    01150          LDX #$13
               01160 .5
45F8: BD DD 4A 01170          LDA L1MSG,X
45FB: 9D 90 71 01180          STA LINE1,X
45FE: BD F1 4A 01190          LDA L2MSG,X
4601: 9D A4 71 01200          STA LINE2,X
4604: A9 00    01210          LDA #$00
4606: 9D B8 71 01220          STA LINE3,X
4609: 9D CC 71 01230          STA LINE4,X
460C: CA       01240          DEX
460D: 10 E9    01250          BPL .5
460F: 60       01260          RTS
               01270 ;
               01280 *
               01290 LOOKAHEAD
4610: AD 73 4B 01300          LDA NEWH    ; GET HORIZ. POS.
4613: 29 07    01310          AND #$07    ; GET # OF BITS INTO SCREEN CHR
4615: 8D 5F 4B 01320          STA XOFF    ; SAVE IT
```

```
4618: AD 74 4B 01330          LDA NEWV
461B: 29 07    01340          AND #$07    ; GET # OF SCAN LINES DOWN INTO SCREEN CHR
461D: 8D 60 4B 01350          STA YOFF    ; SAVE IT
4620: 38       01360          SEC
4621: AD 73 4B 01370          LDA NEWH
4624: E9 30    01380          SBC #$30    ; -LEFT EDGE
4626: 4A       01390          LSR
4627: 4A       01400          LSR
4628: 4A       01410          LSR         ; /8
4629: 8D 5D 4B 01420          STA CHRX
462C: 38       01430          SEC
462D: AD 74 4B 01440          LDA NEWV
4630: E9 20    01450          SBC #$20    ; -TOP EDGE
4632: 4A       01460          LSR
4633: 4A       01470          LSR
4634: 4A       01480          LSR         ; /8
4635: 8D 5E 4B 01490          STA CHRY
               01500 *
               01510 *
4638: A2 14    01520          LDX #$14    ; 20 BYTES/ROW
463A: 20 CC 46 01530          JSR MULTIPLY
463D: 18       01540          CLC
463E: AD 5D 4B 01550          LDA CHRX
4641: 6D 59 4B 01560          ADC RESULT
4644: 85 F0    01570          STA CHRLO
4646: A9 70    01580          LDA /SCREEN
4648: 6D 5A 4B 01590          ADC RESULT+1
464B: 85 F1    01600          STA CHRHI
464D: 20 A5 46 01610          JSR DOCPTR
4650: A9 00    01620          LDA #$00
4652: 8D 65 4B 01630          STA DPOINTER
               01640 LABEL
4655: AC 60 4B 01650          LDY YOFF
4658: B1 F2    01660          LDA (DATAP1),Y
465A: 8D 63 4B 01670          STA BYTE
465D: B1 F4    01680          LDA (DATAP2),Y
465F: 8D 64 4B 01690          STA BYTE+1
4662: AC 5F 4B 01700          LDY XOFF
4665: F0 09    01710          BEQ .2
               01720 .1
4667: 0E 64 4B 01730          ASL BYTE+1  ;COMBINE TWO BYTES
466A: 2E 63 4B 01740          ROL BYTE
466D: 88       01750          DEY
466E: D0 F7    01760          BNE .1
               01770 .2
4670: AC 65 4B 01780          LDY DPOINTER
4673: B1 F7    01790          LDA (POTMP1),Y ; GET BYTE OF TANK DATA
4675: 2D 63 4B 01820          AND BYTE       ; MASK IT WITH OVERLAPPING 8 BITS OF SCREEN
4678: D0 29    01840          BNE COLLISION ;COLLISION RESULT NON ZERO
               01850 * SET FOR NEXT POSITION
               01860 .20
467A: EE 60 4B 01870          INC YOFF    ; SET FOR NEXT BYTE OF CHARACTER DATA
467D: AD 60 4B 01880          LDA YOFF
4680: C9 08    01890          CMP #$08    ; HAVE WE DGONE THROUGH ALL 8 BYTES OF CHAR DATA
4682: D0 13    01900          BNE .4      ; NO.
4684: A9 00    01910          LDA #$00    ; SET POINTER FOR 1ST BYTE OF CHR BELOW
4686: 8D 60 4B 01920          STA YOFF
4689: 18       01930          CLC
468A: A5 F0    01940          LDA CHRLO
468C: 69 14    01950          ADC #$14    ; WIDTH
468E: 85 F0    01960          STA CHRLO
4690: 90 02    01970          BCC .3
```

```
4692: E6 F1    01980          INC CHRHI   ; GO GET POINTERS TO CHRDATA FOR CHARACTERS ON-
               01990 ;                    ;                              -THE NEXT LINE DOWN
4694: 20 A5 46 02000 .3       JSR DOCPTR
               02010 .4
4697: EE 65 4B 02020          INC DPOINTER      ; SET FOR NEXT BYTE OF TANK DATA
469A: AD 65 4B 02030          LDA DPOINTER
469D: C9 10    02040          CMP #$10          ; DONE ALL 16 BYTES OF TANK DATA
469F: D0 B4    02050          BNE LABEL         ; GO BACK TILL THEY ARE ALL DONE
46A1: 18       02060          CLC               ; SET FLAG FOR POSITION OK.
46A2: 60       02070          RTS
               02080 ;
               02090 ;
               02100 COLLISION
46A3: 38       02110          SEC         ; FLAG POS. NO GOOD
46A4: 60       02120          RTS
               02130 ;
               02140 ;
               02150 ;
               02160 ;
               02170 ;
               02180 ;
               02190 * GET CHR & ADJACENT CHR
               02200 DOCPTR
46A5: A0 01    02210          LDY #$01
46A7: A2 03    02220          LDX #$03
               02230 .1
46A9: A9 00    02240          LDA #$00
46AB: 95 F2    02250          STA DATAP1,X
46AD: B1 F0    02260          LDA (CHRLO),Y
46AF: 29 3F    02270          AND #%00111111    ; MASK OFF COLOR BITS
46B1: 99 61 4B 02280          STA CHR1,Y
46B4: 0A       02290          ASL
46B5: 36 F2    02300          ROL DATAP1,X
46B7: 0A       02310          ASL
46B8: 36 F2    02320          ROL DATAP1,X
46BA: 0A       02330          ASL
46BB: 36 F2    02340          ROL DATAP1,X
46BD: CA       02350          DEX
46BE: 95 F2    02360          STA DATAP1,X
46C0: 18       02370          CLC
46C1: B5 F3    02380          LDA DATAP1+1,X
46C3: 69 60    02390          ADC /CHRSET
46C5: 95 F3    02400          STA DATAP1+1,X
46C7: CA       02410          DEX
46C8: 88       02420          DEY
46C9: 10 DE    02430          BPL .1
46CB: 60       02440          RTS
               02450 ;
               02460 ; 8 BIT MULTIPLY
               02470 ; MULTIPLIES ACC BY X REG.
               02480 ; STORES RESULT (L,H) IN RESULT & RESULT+1
               02490 MULTIPLY
46CC: 8D 5B 4B 02500          STA M1      ; MULTIPLIER
46CF: 8E 5C 4B 02510          STX M2      ; MULTIPLICAND
46D2: A9 00    02520          LDA #$0
46D4: 8D 59 4B 02530          STA RESULT
46D7: 8D 5A 4B 02540          STA RESULT+1
46DA: A2 08    02550          LDX #$08
               02560 .1
46DC: 0E 59 4B 02570          ASL RESULT
46DF: 2E 5A 4B 02580          ROL RESULT+1
46E2: 0E 5C 4B 02590          ASL M2
```

```
46E5: 90 0F    02600          BCC .2
46E7: 18       02610          CLC
46E8: AD 59 4B 02620          LDA RESULT
46EB: 6D 5B 4B 02630          ADC M1
46EE: 8D 59 4B 02640          STA RESULT
46F1: 90 03    02650          BCC .2
46F3: EE 5A 4B 02660          INC RESULT+1
               02670 .2
46F6: CA       02680          DEX
46F7: D0 E3    02690          BNE .1
46F9: 60       02700          RTS
               02710 ;
               00140          .IN "D:TANKDATA.SRC"
               00010 * TANK DATA
               00020 * THIS IS D:TANKDATA.SRC
               00030 NEWSET
46FA: 00 00 00
46FD: 00 00 00
4700: 00 00    00040          .HS 0000000000000000
4702: 00 00 00
4705: 1F 1F 18
4708: 18 18    00050          .HS 0000001F1F181818
470A: 00 00 00
470D: F8 F8 18
4710: 18 18    00060          .HS 000000F8F8181818
4712: 18 18 18
4715: 1F 1F 00
4718: 00 00    00070          .HS 1818181F1F000000
471A: 18 18 18
471D: F8 F8 00
4720: 00 00    00080          .HS 181818F8F8000000
4722: 18 18 18
4725: 18 18 18
4728: 18 18    00090          .HS 1818181818181818
472A: 00 00 00
472D: FF FF 00
4730: 00 00    00100          .HS 000000FFFF000000
4732: 00 00 00
4735: 00 00 00
4738: 00 00    00110          .HS 0000000000000000
473A: 00 7E 5C
473D: 2A 66 5E
4740: 6E 00    00120          .HS 007E5C2A665E6E00
4742: 00 7E 7E
4745: 7E 7E 7E
4748: 7E 00    00130          .HS 007E7E7E7E7E7E00
474A: 00 38 76
474D: 36 48 5E
4750: 2C 00    00140          .HS 00387636485E2C00
4752: 00 3C 7E
4755: 7E 7E 7E
4758: 3C 00    00150          .HS 003C7E7E7E7E3C00
475A: 00 34 34
475D: 34 28 2C
4760: 1C 00    00160          .HS 00343434282C1C00
4762: 00 3C 3C
4765: 3C 3C 3C
4768: 3C 00    00170          .HS 003C3C3C3C3C3C00
476A: 00 00 5E
476D: 22 5C 3E
4770: 00 00    00180          .HS 00005E225C3E0000
4772: 00 00 7E
```

```
4775: 7E 7E 7E
4778: 00 00     00190           .HS 00007E7E7E7E0000
                00200 ;
                00210 MAZE
477A: C1 C6 C6
477D: C6 C6 C6
4780: C6 C6     00220           .HS C1C6C6C6C6C6C6C6
4782: C6 C6 C6
4785: C6 C6 C6
4788: C6 C6     00230           .HS C6C6C6C6C6C6C6C6
478A: C6 C6 C6
478D: C2 C5 00
4790: 00 00     00240           .HS C6C6C6C2C5000000
4792: 00 0D 00
4795: 00 00 00
4798: 00 00     00250           .HS 000D000000000000
479A: 00 00 00
479D: 0D 00 00
47A0: 00 C5     00260           .HS 0000000D000000C5
47A2: C5 00 00
47A5: 00 00 0D
47A8: 00 00     00270           .HS C50000000000D0000
47AA: 00 00 00
47AD: 00 00 00
47B0: 00 0D     00280           .HS 000000000000000D
47B2: 00 00 00
47B5: C5 C5 00
47B8: 00 00     00290           .HS 000000C5C5000000
47BA: 00 0D 00
47BD: 00 00 00
47C0: 00 00     00300           .HS 000D000000000000
47C2: 00 00 00
47C5: 0D 00 00
47C8: 00 C5     00310           .HS 0000000D000000C5
47CA: C5 00 00
47CD: 00 00 0D
47D0: 00 00     00320           .HS C50000000000D0000
47D2: 00 00 00
47D5: 00 00 00
47D8: 00 0D     00330           .HS 000000000000000D
47DA: 00 00 00
47DD: C5 C5 00
47E0: 00 00     00340           .HS 000000C5C5000000
47E2: 00 0D 00
47E5: 0D 00 00
47E8: 00 00     00350           .HS 000D000D00000000
47EA: 00 0D 00
47ED: 0D 00 00
47F0: 00 C5     00360           .HS 000D000D000000C5
47F2: C5 0F 0F
47F5: 0F 00 0D
47F8: 00 0D     00370           .HS C50F0F0F000D000D
47FA: 00 09 09
47FD: 09 0F 0B
4800: 0F 0D     00380           .HS 000909090F0B0F0D
4802: 0F 0F 0F
4805: C5 C5 00
4808: 00 00     00390           .HS 0F0F0FC5C5000000
480A: 00 0D 00
480D: 0D 00 09
4810: 09 09     00400           .HS 000D000D00090909
4812: 00 0D 00
```

```
4815: 0D 00 00
4818: 00 C5     00410           .HS 000D000D000000C5
481A: C5 00 00
481D: 00 00 00
4820: 00 00     00420           .HS C500000000000000
4822: 00 09 09
4825: 09 00 00
4828: 00 00     00430           .HS 0009090900000000
482A: 00 00 00
482D: C5 C5 00
4830: 00 00     00440           .HS 000000C5C5000000
4832: 00 00 00
4835: 00 00 09
4838: 09 09     00450           .HS 0000000000090909
483A: 00 00 00
483D: 00 00 00
4840: 00 C5     00460           .HS 00000000000000C5
4842: C5 00 00
4845: 00 00 00
4848: 00 00     00470           .HS C500000000000000
484A: 00 09 09
484D: 09 00 00
4850: 00 00     00480           .HS 0009090900000000
4852: 00 00 00
4855: C5 C5 00
4858: 00 00     00490           .HS 000000C5C5000000
485A: 00 00 00
485D: 00 00 09
4860: 09 09     00500           .HS 0000000000090909
4862: 00 00 00
4865: 00 00 00
4868: 00 C5     00510           .HS 00000000000000C5
486A: C5 00 00
486D: 00 00 00
4870: 00 00     00520           .HS C500000000000000
4872: 00 09 09
4875: 09 00 00
4878: 00 00     00530           .HS 0009090900000000
487A: 00 00 00
487D: C5 C5 00
4880: 00 00     00540           .HS 000000C5C5000000
4882: 00 00 00
4885: 00 00 09
4888: 09 09     00550           .HS 0000000000090909
488A: 00 00 00
488D: 00 00 00
4890: 00 C5     00560           .HS 00000000000000C5
4892: C5 00 00
4895: 00 0D 00
4898: 00 0D     00570           .HS C50000000D00000D
489A: 00 09 09
489D: 09 00 0D
48A0: 0D 00     00580           .HS 00090909000D0D00
48A2: 00 00 00
48A5: C5 C5 0F
48A8: 0F 0F     00590           .HS 000000C5C50F0F0F
48AA: 0D 00 0F
48AD: 0B 0F 09
48B0: 09 09     00600           .HS 0D000F0B0F090909
48B2: 00 0D 0D
48B5: 00 0F 0F
48B8: 0F C5     00610           .HS 000D0D000F0F0FC5
```

```
48BA: C5 00 00
48BD: 00 0D 00
48C0: 00 0D     00620           .HS C50000000D00000D
48C2: 00 00 00
48C5: 00 00 0D
48C8: 0D 00     00630           .HS 00000000000D0D00
48CA: 00 00 00
48CD: C5 C5 00
48D0: 00 00     00640           .HS 000000C5C5000000
48D2: 0D 00 00
48D5: 00 00 00
48D8: 00 00     00650           .HS 0D000000000000000
48DA: 00 00 0D
48DD: 00 00 00
48E0: 00 C5     00660           .HS 00000D00000000C5
48E2: C5 00 00
48E5: 00 0D 00
48E8: 00 00     00670           .HS C50000000D000000
48EA: 00 00 00
48ED: 00 00 00
48F0: 0D 00     00680           .HS 0000000000000D00
48F2: 00 00 00
48F5: C5 C3 C6
48F8: C6 C6     00690           .HS 000000C5C3C6C6C6
48FA: C6 C6 C6
48FD: C6 C6 C6
4900: C6 C6     00700           .HS C6C6C6C6C6C6C6C6
4902: C6 C6 C6
4905: C6 C6 C6
4908: C6 C4     00710           .HS C6C6C6C6C6C6C6C4
                00720 ; TANK DATA
                00730 TANK0
490A: 00 00 00
490D: C6 38 38
4910: C6 38     00740           .HS 000000C63838C638
4912: 38 C6 38
4915: 38 C6 00
4918: 00 00     00750           .HS 38C63838C6000000
                00760 TANK1
491A: 00 00 00
491D: 10 08 20
4920: 1A 5D     00770           .HS 0000001008201A5D
4922: 3C BA 58
4925: 04 10 08
4928: 00 00     00780           .HS 3CBA580410080000
                00790 TANK2
492A: 00 00 00
492D: AA AA AA
4930: 7C 7C     00800           .HS 000000AAAAAA7C7C
4932: 7C 7C AA
4935: AA AA 00
4938: 00 00     00810           .HS 7C7CAAAAAA000000
                00820 TANK3
493A: 00 00 00
493D: 08 10 04
4940: 58 BA     00830           .HS 00000008100458BA
4942: 3C 5D 1A
4945: 20 08 10
4948: 00 00     00840           .HS 3C5D1A2008100000
                00850 TANK4
494A: 00 00 00
494D: C6 38 38
```

```
4950: C6 38     00860           .HS 000000C63838C638
4952: 38 C6 38
4955: 38 C6 00
4958: 00 00     00870           .HS 38C63838C6000000
                00880 TANK5
495A: 00 00 00
495D: 10 08 20
4960: 1A 5D     00890           .HS 0000001008201A5D
4962: 3C BA 58
4965: 04 10 08
4968: 00 00     00900           .HS 3CBA580410080000
                00910 TANK6
496A: 00 00 00
496D: 55 55 55
4970: 3E 3E     00920           .HS 0000005555553E3E
4972: 3E 3E 55
4975: 55 55 00
4978: 00 00     00930           .HS 3E3E555555000000
                00940 TANK7
497A: 00 00 00
497D: 08 10 04
4980: 58 BA     00950           .HS 00000008100458BA
4982: 3C 5D 1A
4985: 20 08 10
4988: 00 00     00960           .HS 3C5D1A2008100000
                00970 TANKTAB
498A: 0A 49 1A
498D: 49 2A 49
4990: 3A 49     00980           .DA TANK0,TANK1,TANK2,TANK3
4992: 4A 49 5A
4995: 49 6A 49
4998: 7A 49     00990           .DA TANK4,TANK5,TANK6,TANK7
                01000 TURRET0
499A: 00 3C 18
499D: 18 18 18
49A0: 18 3C     01010           .HS 003C18181818183C
49A2: 3C 18 00
49A5: 00 00 00
49A8: 00 00     01020           .HS 3C18000000000000
                01030 TURRET1
49AA: 00 00 02
49AD: 03 06 0E
49B0: 1C 3C     01040           .HS 00000203060E1C3C
49B2: 3C 18 00
49B5: 00 00 00
49B8: 00 00     01050           .HS 3C18000000000000
                01060 TURRET2
49BA: 00 00 00
49BD: 00 00 00
49C0: 19 3F     01070           .HS 000000000000193F
49C2: 3F 19 00
49C5: 00 00 00
49C8: 00 00     01080           .HS 3F19000000000000
                01090 TURRET3
49CA: 00 00 00
49CD: 00 00 00
49D0: 18 3C     01100           .HS 000000000000183C
49D2: 3C 1C 0E
49D5: 06 03 02
49D8: 00 00     01110           .HS 3C1C0E0603020000
                01120 TURRET4
49DA: 00 00 00
```

```
49DD: 00 00 00
49E0: 18 3C     01130           .HS 000000000000183C
49E2: 3C 18 18
49E5: 18 18 18
49E8: 3C 00     01140           .HS 3C18181818183C00
                01150 TURRET5
49EA: 00 00 00
49ED: 00 00 00
49F0: 18 3C     01160           .HS 000000000000183C
49F2: 3C 38 70
49F5: 60 C0 40
49F8: 00 00     01170           .HS 3C387060C0400000
                01180 TURRET6
49FA: 00 00 00
49FD: 00 00 00
4A00: 98 FC     01190           .HS 00000000000098FC
4A02: FC 98 00
4A05: 00 00 00
4A08: 00 00     01200           .HS FC98000000000000
                01210 TURRET7
4A0A: 00 00 40
4A0D: C0 60 70
4A10: 38 3C     01220           .HS 000040C060703830
4A12: 3C 18 00
4A15: 00 00 00
4A18: 00 00     01230           .HS 3C18000000000000
                01240 TURTAB
4A1A: 9A 49 AA
4A1D: 49 BA 49
4A20: CA 49     01250           .DA TURRET0,TURRET1,TURRET2,TURRET3
4A22: DA 49 EA
4A25: 49 FA 49
4A28: 0A 4A     01260           .DA TURRET4,TURRET5,TURRET6,TURRET7
                01270 ;
                01280 NDLIST
4A2A: 70 70 70
4A2D: 46        01290           .HS 70707046
4A2E: 00 70     01300           .DA SCREEN
4A30: 06 06 06
4A33: 06 06 06
4A36: 06 06     01310           .HS 0606060606060606
4A38: 06 06 06
4A3B: 06 06 06
4A3E: 06 06     01320           .HS 0606060606060606
4A40: 06 06 86
4A43: 46        01330           .HS 06068646
4A44: 90 71     01340           .DA WINDOW
4A46: 06 06 06
4A49: 41        01350           .HS 06060641
4A4A: 2A 4A     01360           .DA NDLIST
                01370 ;
                01380 DOFFS
4A4C: 00 00 00
4A4F: 00 00 01
4A52: 01 01     01390           .HS 0000000000010101
4A54: 00 FF FF
4A57: FF 00 00
4A5A: 00 00     01400           .HS 00FFFFFF00000000
                01410 HOFFS
4A5C: 00 01 01
4A5F: 01 00 FF
4A62: FF FF     01420           .HS 0001010100FFFFFF
```

```
                    01430 VOFFS
4A64: FF FF 00
4A67: 01 01 01
4A6A: 00 FF         01440           .HS FFFF0001010100FF
                    01450 ;
                    01460 COLORS
4A6C: 44 94 0E
4A6F: 0E 16 46
4A72: 96 4E         01470           .HS 44940E0E1646964E
4A74: 00            01480           .HS 00
                    01490 PLAYTAB
4A75: 66 67         01500           .DA /PLAYER2,/PLAYER3
                    01510 PLAYTAB2
4A77: 64 65         01520           .DA /PLAYER0,/PLAYER1
                    01530 SHELLS
4A79: 01 04         01540           .HS 0104
                    01550 SHTMSK
4A7B: FE FB         01560           .HS FEFB
                    01570 HITMASK
4A7D: 04 08         01580           .HS 0408
                    01590 WHO
4A7F: 01 00         01600           .HS 0100
                    01610 UZAP
4A81: 00 00 00
4A84: 00 00 00
4A87: 00 00 00 01620            .HS 000000000000000000
4A8A: 00 00 00
4A8D: 00 00 00
4A90: 00 00 00 01630            .HS 000000000000000000
                    01640 DZAP
4A93: 00 00 00
4A96: 00 00 00
4A99: 00 00 00 01650            .HS 000000000000000000
4A9C: 00 00 00
4A9F: 00 00 00
4AA2: 00 00 00 01660            .HS 000000000000000000
                    01670 UZTAB
4AA5: 81 4A 8A
4AA8: 4A            01680           .DA UZAP,UZAP+9
                    01690 DZTAB
4AA9: 93 4A 9C
4AAC: 4A            01700           .DA DZAP,DZAP+9
                    01710 IDLE
4AAD: E0 A0         01720           .HS E0A0
                    01730 VROOM
4AAF: 80 60         01740           .HS 8060
                    01750 STARTH
4AB1: 40 B0         01760           .HS 40B0
                    01770 STARTV
4AB3: 60 80         01780           .HS 6080
                    01790 SMSG
4AB5: 00 00 00
4AB8: 00 00 00
4ABB: 00 00         01800           .AT "        "
4ABD: 13 1A 10
4AC0: 10 00 00
4AC3: 00 00         01810           .AT "3:00    "
4AC5: 00 00 00
4AC8: 00            01820           .AT "    "
                    01830 EMSG
4AC9: 00 00 00
4ACC: 00 00 00
```

```
4ACF: 27 21     01840           .AT "      GA"
4AD1: 2D 25 00
4AD4: 2F 36 25
4AD7: 32 00     01850           .AT "ME OVER "
4AD9: 00 00 00
4ADC: 00        01860           .AT "    "
4ADD: 00 30 2C
4AE0: 21 39 25
4AE3: 32 11     01870 L1MSG     .AT " PLAYER1"
4AE5: 00 00 00
4AE8: 00 30 2C
4AEB: 21 39     01880           .AT "    PLAY"
4AED: 25 32 12
4AF0: 00        01890           .AT "ER2 "
                01900 L2MSG
4AF1: 00 00 00
4AF4: 00 00 10
4AF7: 00 00     01910           .AT "     0  "
4AF9: 00 00 00
4AFC: 00 00 00
4AFF: 00 00     01920           .AT "        "
4B01: 10 00 00
4B04: 00        01930           .AT "0   "
                01940 SCPOS
4B05: 04 0F     01950           .HS 040F
                01960 TDLIST
4B07: 70 70 70
4B0A: 70 70 70
4B0D: 70 70     01970           .HS 7070707070707070
4B0F: 70 70 46  01980           .HS 707046
4B12: 1B 4B     01990           .DA TITLED
4B14: 70 06 70
4B17: 06 41     02000           .HS 7006700641
4B19: 07 4B     02010           .DA TDLIST
                02020 TITLED
4B1B: 00 00 00
4B1E: 00 00 34
4B21: 21 2E     02030           .AT "     TAN"
4B23: 2B 00 22
4B26: 21 34 34
4B29: 2C 25     02040           .AT "K BATTLE"
4B2B: 00 00 00
4B2E: 00        02050           .AT "    "
4B2F: 80 80 A1
4B32: 80 B4 B7
4B35: AF 80     02060           .AT -"  A TWO "
4B37: B0 AC A1
4B3A: B9 A5 B2
4B3D: 80 A7     02070           .AT -"PLAYER G"
4B3F: A1 AD A5
4B42: 80        02080           .AT -"AME "
4B43: 80 80 80
4B46: 80 E2 F9
4B49: 80 E4     02090           .AT -"    by d"
4B4B: E1 EE 80
4B4E: F0 E9 EE
4B51: E1 EC     02100           .AT -"an pinal"
4B53: 80 80 80
4B56: 80        02110           .AT -"    "
                02120 CSHIFT
4B57: 50 90     02130           .HS 5090
                02140 ;
```

```
                02150 ;
                02160 ;
                00150          .IN "D:TANKVAR.SRC"
                00010 * TANK GAME VARIABLE
                00020 * THIS IS D:TANKVAR.SRC
                00030 FIRSTVAR
4B59:           00040 RESULT   .BS 2 ; PRODUCT
4B5B:           00060 M1       .BS 1 ; MULTIPLIER
4B5C:           00070 M2       .BS 1 ; MULTIPLICAND
4B5D:           00080 CHRX     .BS 1 ; HORZ. POS. OF CHAR TANK IS ON TOP OF
4B5E:           00090 CHRY     .BS 1 ; VERT. POS. OF CHAR TANK IS ON TOP OF
4B5F:           00100 XOFF     .BS 1 ; HOW MANY PIXELS TANK IS OFFSET FROM LEFT EDGE OF CHR
4B60:           00110 YOFF     .BS 1 ; HOW MANY SCAN LINES TANK IS OFFSET FROM TOP OF CHR
4B61:           00120 CHR1     .BS 1 ; CHAR TANK IS ON TOP OF
4B62:           00130 CHR2     .BS 1 ; ADJACENT CHARACTER TANK MAY BE ON TOP OF
4B63:           00140 BYTE     .BS 2 ; ACTUAL 8 BITS OF SCREEN DATA TANK IS ON TOP OF
4B65:           00150 DPOINTER .BS 1 ; DATA POINTER INTO TANK DATA
4B66:           00160 SAVEX    .BS 1 ; TEMP. X REG. STORAGE
4B67:           00170 SAVEY    .BS 1 ; TEMP. Y REG. STORAGE
4B68:           00180 TANK     .BS 1 ; WHICH TANK IS BEING DONE
4B69:           00190 GAME     .BS 1 ; GAME IN PROGRESS FLAG
4B6A:           00200 ENABLE   .BS 1 ; FLAG FOR VBI TO SKIP CODE IF GAME NOT RUNNING
4B6B:           00210 PLAYERH  .BS 2 ; TANK HORIZ. POS.
4B6D:           00220 PLAYERV  .BS 2 ; TANK VERT. POS.
4B6F:           00230 PLAYERD  .BS 2 ; TANK DIRECTION OF MOVEMENT
4B71:           00240 PLAYERF  .BS 2 ; DIRECTION TANK IS FACING
4B73:           00250 NEWH     .BS 1 ; TEMPORARY COPIES OF TANK POSITIONS
4B74:           00260 NEWV     .BS 1 ;  "
4B75:           00270 NEWD     .BS 1 ;  "
4B76:           00280 NEWF     .BS 1 ;  "
4B77:           00290 NEWTD    .BS 1 ;  "
4B78:           00300 TEMP0    .BS 1 ; TEMPORARY STORAGE
4B79:           00310 TEMP1    .BS 1 ;  "
4B7A:           00320 TEMP2    .BS 1 ;  "
4B7B:           00330 TEMP3    .BS 1 ;  "
4B7C:           00340 TEMP4    .BS 1 ;  "
                00350 ;
4B7D:           00360 TURRETF  .BS 2 ; FLAG FOR TURRET MODE
4B7F:           00370 TURRETD  .BS 2 ; TURRET DIRECTION
                00380 ;
4B81:           00390 SHOTS    .BS 2 ; SHOT SOUND FLAG
4B83:           00400 SHOTD    .BS 2 ; SHOT DIRECTION
4B85:           00410 SHOTH    .BS 2 ; SHOT HORIZ. POS.
4B87:           00420 SHOTV    .BS 2 ; SHOT VERT. POS.
4B89:           00430 SHOTF    .BS 2 ; SHOT IN PROGRESS FLAG
4B8B:           00440 CYCLES   .BS 2 ; EXPLOSION SEQUENCE FLAG AND COUNTER
4B8D:           00450 INDEX    .BS 1 ; AN INDEX
4B8E:           00460 IMMUNE   .BS 2 ; TEMPORARY TANK IMMUNITY TIMER
4B90:           00470 TIMER    .BS 1 ; JIFFY COUNTER
4B91:           00480 MINUTES  .BS 1 ; TIME
4B92:           00490 SECONDS  .BS 1 ; TIME
4B93:           00500 SCORE    .BS 2 ; PLAYER SCORES
                00510 ;
                00520 ;
                00530 LASTVAR
                00540 ;
                00160 *
```

CHAPTER 10

GAME DESIGN THEORY

There is no sure-fire way to predict whether a game will be successful, but there are certain attributes that contribute to success. Certainly a game should have a goal—without one, what is the point in playing? Rules should be straightforward and logical. The game should also be a challenge; if it requires no skill, you will quickly tire of it. A game should evoke either fantasy or your innate curiosity; if it isn't novel or puzzling, it becomes boring. And lastly, arcade games, that by definition have a lot of action, should be easily controllable.

Game objectives take two different forms. There are games where you gradually approach the goal, like destroying a fleet of invaders in *Galaxian*, eating all the dots in *Pac Man*, or rescuing all of the hostages in *Choplifter*. There are other games where the goal is to avoid catastrophe. Examples of this range from preventing a nuclear power plant meltdown in *Scram*, to saving your cities during a nuclear missile attack in *Missile Command*, or preserving all of your fuel canisters in *Ripoff*. Do not confuse these two kinds of game objectives with the simplistic and mindless act of scoring lots of points by shooting everything that moves.

Goals must suit a player's expectations or fantasies. This is why certain people like certain types of games better than others. The battle lines of good against evil lurk in the background of many space games, wherein evil, menacing invaders are bent on the destruction of Earth. It becomes the player's goal to protect the Earth as long as possible while scoring the most points for killing aliens. Other appealing goals range from accumulating the most treasure while exploring a dangerous cavern, to escaping from a crumbling building before it collapses or your food runs out.

Computer game fantasies derive some of their appeal from the emotional needs they satisfy. Different fantasies appeal to different people. Sometimes the fantasy is simply an adolescent emotional release as in *Food Fight*, where you battle pie-throwing chefs with tables full of messy food. The fantasy of destroying objects during a game appeals to others. It can take the form of popping balloons by bouncing a clown off a teeter-totter, as in *Clowns and Balloons*, or breaking out bricks in a wall, as in *Breakout*. In each case, the partially-destroyed wall or rows of balloons presents a visually compelling goal and a graphic scorekeeping device as well.

Goals in most games imply an end point, either when the goal is reached or when you fail. It is often important to make sure the game doesn't just go on and on forever. Limits should be set. Sometimes these take the form of time limits or constantly diminishing amounts of ammunition, balls, or ships. The most wides-

pread limiting factor, at least on home computers, is speeding up the game. It is also the most abused. A game tempo, where it is neither humanly nor mechanically possible to withstand the onslaught of the computer's forces, cheats the player.

For a game to be considered challenging, it should have a goal where the outcome is uncertain. If the player is certain to reach the goal or certain not to reach it, the game is unlikely to present a challenge, and the player will lose interest. It is very easy to introduce randomness into the game either by hiding important information or by introducing random variables that draw the player toward disaster. Be careful not to overdo this, since a totally random game lacks a skill factor. Players quickly discover that they have no control over the outcome.

One of the more important design elements in any game is a logical set of rules. The rules can be extremely simple or utterly complex, but they must make sense. Since the game must follow its theme, any rules or variations should stem directly from that theme. It is pointless to throw in game elements that simply don't belong just because you think that confusing the player would make the game more difficult. For instance, *Donkey Kong*, one of the best jumping, climbing arcade games, doesn't require the player to shoot everything in sight, just avoid obstacles to reach the goal. Similarly, a tough, shoot-'em-up game like *Galaxian* keeps its fluid alien attack uncluttered by distracting game elements.

A game like *Galaxian* is considered asymmetric. It is not a balanced game because both the player and the computer's alien fleet are unbalanced in strength. Yet the differences between the advantages and disadvantages of the two opponents are too similar to build triangular relationships that make an arcade game more interesting.

The triangular relationship is one in which each opponent can defeat one other or be defeated by the third. The relationship is often used in many games to lure the player into a trap by sacrificing a weakly-armed player. *Battlezone* is a good example. The computer maneuvers the saucer to entice the human into a poor position against the tank. *Time Pilot* lures you into a poor position more subtly by placing the bonus parachute directly in line with the incoming enemy fighter pack. Other games use the bonus to distract you. *Sky Blazer* nearly always drops a fuel canister just at the time that your target appears, and bomb targets come up in *Xevious* just when you are engaged in a heavy fire fight with the alien armada.

A variable difficulty level is often used to alter the game's level of play. These levels, often with ego-satisfying names like *Star Commander* or *Pilot*, can be set by the player. Many games are designed to become harder the further you progress. The increasing skill level requirement presents an added challenge, while preventing the player from growing complacent. Often the technique is to speed up the game or place additional enemy craft into battle. The player is required to play faster and better, honing his reflexes during the process. Another variation allows less time to complete your objective as the difficulty increases.

Care must be taken so that the game's level of difficulty progresses evenly from beginner to expert level. Players' scores should reflect a steady improvement in what is known as a positive monotonic curve. A game with a relatively flat curve is hard to

learn, while a sharp jump means that there is some trick required to master the level. Games that don't have a positive monotonic curve frustrate players because they fail to provide reasonable opportunities to better one's score.

Any good game should offer a reward for reaching increasingly difficult levels of play. Often, bonus points, extra balls, ships, or more ammunition are rewarded for exceeding score thresholds. It is important that the rewards for winning outweigh the disappointment of losing. A player's ego is involved. A person wants to beat a challenging game, not be humiliated each time he loses.

The ideal arcade game should foster the illusion of winnability at all levels of play. One important factor is a clean and simple game design. Too much detail or too many rules may intimidate the player. If a player believes that his failure was caused by a flaw in an overly complex game or by the controls, he will consider the game unfair and quit. On the other hand, if a player perceives failure to be attributed to correctable errors on his part, then he believes the game to be winnable and will play repeatedly to master the game. It's as if the player teases himself to play one more time.

Appealing to a player's curiosity effectively keeps a game interesting. While novelty is sometimes a crucial factor in the original purchase, if the game has little depth, it becomes repetitious and boring. One method that appeals to many game designers is to have the game progress to slightly different scenarios. Some games change the opposition, while others vary the scenery; some do both. The player has to excel if he is to satisfy his curiosity. Games like *Threshold,* which progresses through twenty-four sets of alien spacecraft, or *Vanguard,* in which both the scenery and alien craft changes, offer strong curiosity incentives.

These spurs to a player's interest in the game are called "Perks." They are most important just when the player thinks he has the game figured out. Perks must be carefully timed so that the player does not give up on the game because not enough happens soon enough or because everything the game has to offer has been seen too soon. The most common perk is an extra life. Consider these coin-op video games: *Pac Man, Donkey Kong, Dig Dug, Joust, Mario Bros.,* and *Tempest.* These games have multiple screens. The different screens by themselves are a perk, but what these games have in common is the time at which an extra life is rewarded. The extra lives generally come to the average player at some point in his third screen. This is hardly coincidental. The screens are scheduled at a specific rate, somewhat dependent on the player's skill. The extra life on the third screen comes in just before the average player might become exasperated and so not put in another quarter. The novice player is usually out of lives at this point, too.

Some games use cartoon intermissions to perk up the game. The player's interest is renewed with each cartoon. For many players, seeing the next cartoon becomes a personal goal. Placing hidden features not even hinted at in the rules is another clever perk. These embellishments are left for the experienced player to discover. They can even brighten up the earlier levels of a game which has become dull for the expert. For example, in the coin-op version of *Star Wars,* players hear the voice of Obi-Wan Kenobi admonishing them to use the Force. Nothing in the game instruc-

tions tell them what to do. Only by experimentation will a player realize that he must fly through the trench without firing a shot to receive a substantial point bonus upon reaching the exhaust port. The high score feature can also be considered as a "perk." While it doesn't renew interest within the game, it is important because it can renew interest in playing even a mediocre game again. The high score itself presents a personal goal to reach, whether it be to beat your own high score or someone else's.

Sometimes varying the player's emotional response during the game serves as a "perk." Tension can be relieved during a tense shoot-'em-up with occasional comic relief, while cute games sometimes need touching moments. Remember games as entertainment need contrast. In sum, the varieties of perks are endless, but their objective is the same: renew interest in the game before the player becomes tired of it.

A game's controllability is one of the more important considerations in design. It is sometimes referred to as human engineering. Designers usually choose between keyboard and paddle/joystick control. While eye/hand coordination is more effective with paddles or joysticks, programmers attempting to create games with too many control functions will opt for a keyboard control system. At times, they produce a game that requires nine or ten keyboard controls which, unfortunately, only a pianist can operate. Games whose controls require considerable time to master often prove frustrating to play.

The ability to accurately control the action is crucial in the design. If the screen does not respond immediately to the player's input, the player may end up feeling out of sync and become frustrated with the game. The programmer must insure through careful design that the game responds properly to the player. The player shouldn't believe, for example, that the computer made a wrong turn for him in a maze game.

Apparently, Atari owners like games which pit them against a competitive computer opponent. In several multi-player games, groups of two or more simultaneously compete against each other. Most of these contests are sports or card games involving two to four players. The cooperative game is rarely seen, except in cases where the computer competitor is much too skillful. The arcade game *Ripoff* involves a computer opponent that is more than a match for two players playing simultaneously. The battle is so fast and fierce that the teammate's ship has to be protected from his partner's bullets. The home computer version of *Wizard of Wor* offers a choice of competitive or cooperative play. It is a tough game that really needs cooperative play if the more advanced game levels are to be reached, but cooperation in this game is by agreement, not by mutual invulnerablity. Your partner is even worth 1000 points if you mistakenly blunder. It is quite interesting to watch cooperation turn into a fiercely competitive game after one player inadvertently walks into the other player's line of fire.

So far, we have discussed theory and generalizations that should increase a game's playability and appeal to the public. Concrete examples of the more popular games should give you a much more solid foundation for your own designs.

Example Arcade Games

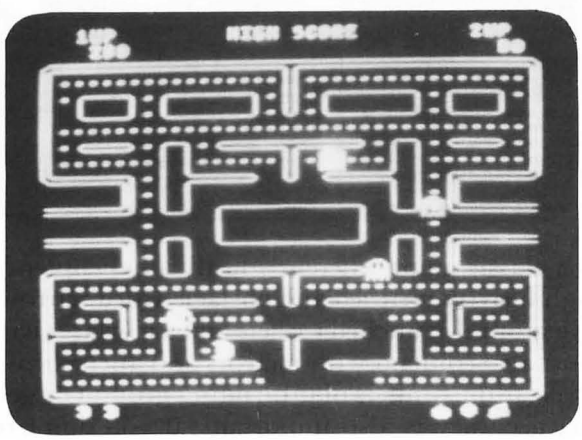
Space Invaders was the first really popular arcade game. The object is to defend your turf against an alien horde of ferocious invaders who attack your castles and gun bases with a barrage of undulating bullets. It is actually a timed game, since you only have a limited period to destroy the entire attacking wave before they descend to the ground in marching formations and overrun your lone gun base.

The elimination of each alien acts as a visual scorekeeping device. You can never win, only survive as long as possible (thus getting the maximum playtime for your quarter). Elimination of each attacking wave, however, is an intermediate goal that staves off your inevitable doom. Each successive level becomes more difficult since the aliens, which begin their attack increasingly closer to Earth each round, limit the amount of time that you have to destroy them. Their constant approach to your mobile gun base decreases the reaction time needed to avoid enemy fire.

Shoot-'em-up games like *Sneakers, Galaxian, Threshold,* and *Galaga* are actually spin-offs of the *Space Invaders* theme. Whether they are set in space or on the ground, each has a variety of targets bent on your destruction. The targets or attackers are no longer static. Either they appear to dodge your fire, or they resort to kamikaze-type attacks.

The strong appeal of these types of games is based on curiosity and game depth. You are inspired to do better with each game just to see what the attackers are going to look like in the next level and what their tactics might be. The design goal is variety, with each successive level slightly harder than the last. Although most offer an unlimited number of bullets, *Threshold* controls rapid, random, and wasteful firing by overheating your lasers. Thus, your firing must be more accurate and paced during the game.

The popularity of *Pac Man* can be attributed to the game's design. First, it satisfies the fantasy concept of a person's childhood dreams. As children, we dreamt that we were being chased by evil monsters or ghosts, and we felt powerless to stop them. We

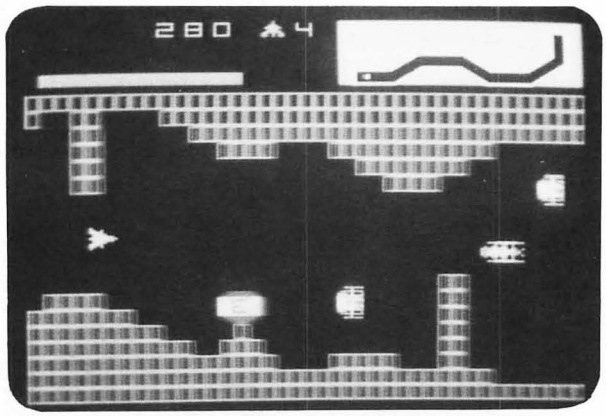


PAC MAN

wished that there were some way to turn the tables, if only for a few moments. *Pac Man's* four energy dots fulfill that fantasy. The game also offers the visual feedback of the number of remaining dots to be eaten at each level. And since clearing each individual level is an immediate goal, even beginners believe a level can be cleared. Because *Pacman* is a game of consumption rather than one of destruction, it appeals to players of both sexes.

The game becomes a learning experience for the more advanced player, since the ghosts follow a discernible, non-random pattern. A player is eventually able to predict their movements and, consequently, to develop a technique to clear all of the dots on a particular level. The long term goal is survival and the highest score. The game is designed so that you gain more pleasure as you get better. Thus, players are willing to devote the time and money to master the game.

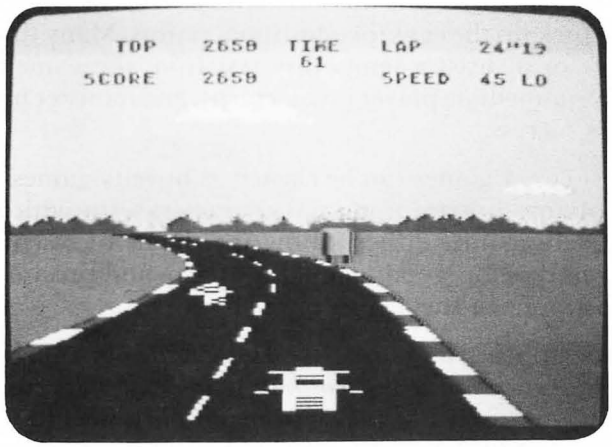
Scrolling games, such as *Super Cobra, Vanguard,* and *Tail of Beta Lyrae,* wherein your ship travels over a multi-screen world, benefit strongly from player curiosity and visual variety. *Vanguard,* a shoot-'em-up game in which a variety of enemy vessels and creatures attack your ship, has an extremely long sinuous tunnel with various types of chambers. The game has so many sections, combined with scrolling directions which change from horizontal to diagonal to vertical, that it is like playing many different arcade games at once. The player is given the option several times during the game to enter battle with a time-limited, energized spacecraft equipped for ramming the enemy, or merely four plain, old directional lasers. A map displayed in the lower corner informs the player of his progress. The curiosity factor is so enticing in this game that thirty seconds are provided to lure you into inserting another quarter in order to continue from where you left off.



VANGUARD

Super Cobra is a classic game wherein you fly a helicopter over scrolling alien terrain and through heavily fortified and obstacle-filled narrow tunnels. Initially, you have to survive ground-launched rockets and a few laser bases, but as the game progresses you must also contend with meteors and alien ships. Using either your bombs or lasers to clear the tunnels of protruding ground targets is crucial to your survival. Bombing accuracy is also important. If you don't replenish your fuel supply by hitting enough fuel depot targets, your game will soon be over.

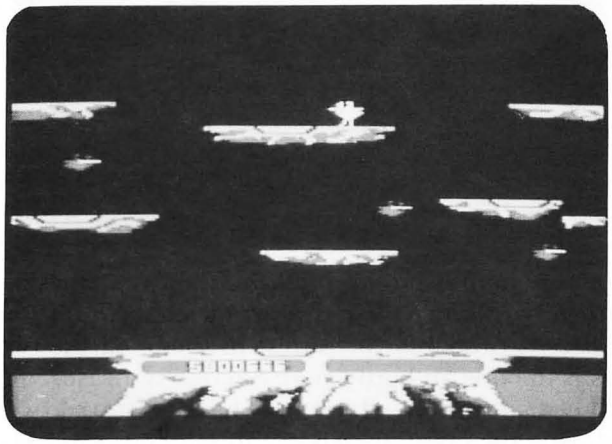
Pole Position, a highly competitive game, appeals to many players because it mixes just the right amount of fantasy with reality. It fulfills the fantasy of being a race car driver without the inherent danger. Crashes are never fatal and do not end the game. The goal of the game is to qualify for and complete the race. In a sense, it is a very realistic simulation requiring shifting gears and precise steering on a scrolling roadway. The player has a three-dimensional view of the course and his car, as if he were following it from fifty feet behind, a sort of out-of-body effect.



POLE POSITION

Joust immediately comes to mind when discussing a pure fantasy game that traces its roots to the glamorous days of medieval chivalry. Instead of presenting two knights in shining armor dueling on horseback, *Joust* allows the player to fly his ostrich-like mount to do battle in midair. The player does not shoot his opponents but defeats them by ramming his mount and lance into theirs, sometimes delicately, sometimes violently. The higher mount always wins. The player gains the excitement of physical contact without a bloody nose.

The game constantly forces the player into action. He must keep hitting the action button to make his mount fly. When the player takes a short rest between screens, his

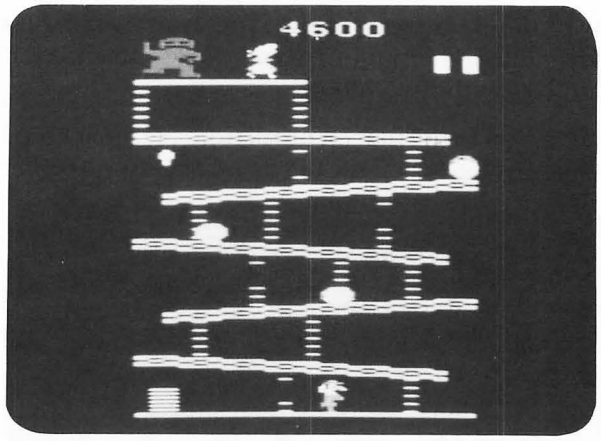


JOUST

surrogate also rests and does not continue to fly along aimlessly as it might in other games. Two players can play simultaneously but are not forced into partnership. The Lava Troll on the bottom of the screen is an additional menace both to the player and his enemies. It attempts to grab at anything close enough and drag it into the lava. This sometimes works to the player's advantage, since the lava can imprison an enemy and make it easier to destroy. A more formidable enemy, the pterodactyl that appears on higher levels, requires the player to discover a way to defeat it. As an added perk, every fifth screen is a bonus level where the player need not fight anyone but simply pick up the eggs for additional points. Many times this earns the player an extra life or at least a temporary rest from the game's pace. Physical contact, originality, immediate player involvement, and monster interaction are key parts of this game's success.

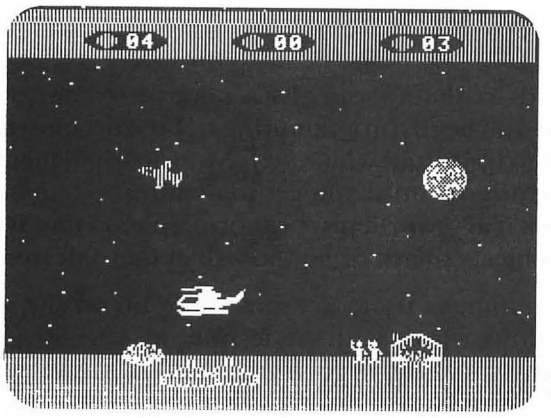
Some of the most clever games can be classed as novelty games. These are often "cute" games involving human or animal characters with which the player can identify. These novelty games either follow the theme of rescuing someone, or require the player to develop good manual dexterity and precise timing skills in order to avoid catastrophe or the demise of the hero.

The rescue theme appears in games like *Donkey Kong, Donkey Kong Junior, Fantasy, and the Adventures of Roby Roto.* In many cases an actual rescue doesn't take place, but the theme carries the player from one portion of the game to the next. In both *Donkey Kong* and *Fantasy,* the girl is whisked away to the next screen just before the player reaches her. The objective isn't the rescue but to overcome the obstacles barring your way. Learning the patterns and precise timing through repeated play hones the player's skill.



DONKEY KONG

Although playing the hero is rare in these games, two games have followed this theme: The *Adventures of Roby Roto* in the arcade, and *Choplifter* on most microcomputers. The latter is probably the purest in theme of the two. The rescue of sixty-four hostages is the one and only goal. Success is measured in the number of hostages rescued. The fact that the player may have destroyed twenty-seven enemy tanks and planes during the mission adds nothing to the score. Thus, while details

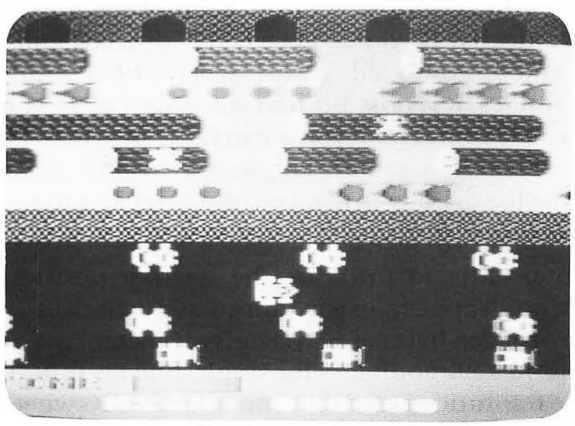


CHOPLIFTER

like the hostage's waving builds empathy for the hostages, the appeal is simply the ego-satisfying role of playing the hero.

In the final group of novelty games, the player must avoid the calamity of losing a life. The goals and obstacles in these games differ widely. *Crazy Climber* requires the player to scale a building while windows close to block the path, and angry tenants, attempting to knock the climber off the building, drop flower pots on his head. *Frogger* has the player brave traffic in a test of precise timing skills. And playing Tarzan in *Jungle Hunt* requires dexterity and timing skills to swing from vine to vine like a trapeze artist, or risk death in the fall. In each of these games the cuteness is what first attracts the audience, but it is the development of the player's timing skills and game depth that keeps him playing. Again, the concept is variety, along with increasing levels of difficulty.

Arcade games have that indefinable ability to make you feel that your losing is just a fluke, and that if you play just one more time, you'll beat it. If you can design a game that is fun and exciting to play, and has that added quality, then you have designed an addictive game, and wealth beyond your wildest dreams may be yours.



FROGGER

What Can Go Wrong

The best piece of advice that we can give a game programmer is to carefully plan out your game before you begin programming it. First decide what results you want and work backwards to figure out what you have to do to get them. If it doesn't work, change your concept or goal until you get something you are satisfied with. Make sure the game follows real—world physical principle, so that it feels right on a gut level. For example, objects smash or bounce when they fall from any height.

Many novice programmers try to get something up on the screen immediately. Actually, there is nothing wrong with this technique. After all, it does give you some encouragement to continue. However, most develop their games on a piecemeal basis, adding something because it looks good or because they need more action. The result is that they soon run out of players or characters and are forced to do a very painful rewrite.

Everyone prefers to organize his game differently. Some, like me, prefer the tight-structured approach of a flowchart; others, like my partner, just write down a rough outline of the order of events in the game. Whichever approach you prefer, we strongly recommend that you develop many of your frequently used routines as independent subroutines. This approach simplifies the logic of the main code loop.

We carefully planned all of the games in this book before we wrote them. This means that we considered where items like screen memory, player-missile memory, the character set, and the actual game code were placed in memory. We roughly flowcharted the game's main logic loop. We then wrote the code in small chunks, but in such manner that it always ran, or at least was supposed to run.

The first priority was to draw the playfield. This generally means that we had to get the display list right and move the character set data into the correct section of memory. It may sound like a piece of cake, but some terrible things can go wrong. Sometimes the display list is too long because you forgot that the first LMS instruction is one of the mode lines. The screen rolls or goes wacko. Maybe you let the display list inadvertently cross a 1K boundary, or allowed screen memory to cross a 4K boundary in the middle of a mode line. Each of these mistakes can cause the screen to behave erratically. If you do get a stable display, and it isn't the one you specified, perhaps you forgot to tell ANTIC where either your display list or your RAM character set resides. It is possible that you didn't place your character set on a 1K boundary, or on a ½K boundary if you are in GR.1 or GR.2. The fastest method to troubleshoot the problem in Assembly language is to enter the monitor and look at the intended areas for the display list and character set and to see if they are actually there. It is quite possible your memory move routine is faulty.

The next step is usually to initialize the starting positions of your players. Sometimes they just don't appear, so you immediately check to see if the player shape is actually in the proper 256 bytes of player-missile memory. You should also check that the PMBASE is on a 2K boundary for single-line resolution players (1K boundary for double-line resolution), and that you told ANTIC where that is. If that isn't the fault, some programmers make the mistake of trying to read a player's horizontal

position by looking at the ANTIC horizontal position register. You may be able to write a horizontal position to the hardware location, but you read collisions from these same hardware locations. If you are going to increment a player's horizontal position, you will need to update a RAM location before writing the value into the hardware register. If this doesn't appear to be the problem, there are two other possibilities. First, you may have forgotten to turn on player-missile graphics switches. But probably the most frequent mistake is to forget to set the player's shadow color register. If it isn't set, it defaults to the background color, blends in with the background and disappears. Always use the shadow color register to change or set color registers, or the change may only last one television frame because the hardware registers are updated every VBlank. The only exception to this rule is when you use a Display List Interrupt to change colors midscreen.

Display List and Vertical Blank Interrupts can sometimes cause unforeseen problems. Inexperience is usually the culprit. The first thing to remember is that you must have a program to interrupt from. Since it is easy to write a simple game entirely in Deferred Vertical Blank, the main loop can be as short as FOREVER JMP FOREVER. The machine will hang up if you don't have somewhere to jump back to at the end of the VBI. On the other hand, if the code is too long, it will be interrupted by the next VBlank before it finishes. Unexpected results, such as a garbaged screen, may occur. The most common problems with Display List Interrupts occur when you forget to save your registers before entering the routine or forget to restore them before exiting. A mistake here will lock up the machine. The other problem is when the interrupt seems to occur on the wrong mode line. Remember that the interrupt has to be set on the mode line before the interrupt is to occur.

BASIC programmers who use Machine language subroutines sometimes encounter strange problems. If you are going to incorporate a VBlank routine, make sure you clear the decimal mode at the beginning. This is especially important if your program uses decimal arithmetic internally. Another problem occurs when you pull the incorrect number of bytes off the stack. This can lock up the machine on the return if the return address on the stack is incorrect. Unfortunately, these subroutines are very difficult to test in Assembly language without constructing a setup routine to simulate the stack environment.

BASIC is usually very forgiving, so it is unlikely that you will lock up the machine if you aren't using Machine language subroutines. One of the most common display mistakes is forgetting to set a graphics mode after you lower RAMTOP to reserve space for your RAM character set and player-missile graphics. If you forget, you will still have a Graphics 0 display just below the old RAMTOP. The new graphics call will actually place the screen below RAMTOP.

We hope we have suggested adequate solutions for the most common errors that might occur in your games. We have learned many of these by bitter and frustrating experience. We will admit that these weren't the only errors that we encountered when programming the code in this book. However, most of the others were logic problems that one of us alone couldn't trace. For example, when I was programming the maze game, I programmed the manual mode for the joystick-controlled letter

first. It worked fine, but became buggy when I added the auto mode. Sometimes the letter would behave properly, yet at other times it would escape the maze walls. I single-stepped the code repeatedly and the legal move flags were always set correctly. Days went by and I couldn't find any cause for the anomaly. My partner discovered that it only happened just after the stick was returned to neutral. While legal moves were reset at the center of each maze block, moving the stick was required to close pathways other than those in the direction of movement or in reverse. Nothing was done in the neutral position because the letter was stopped in the non-auto mode. Since I had neglected to close gates when I was in the neutral position, the joystick-controlled letter was now traveling in some direction automatically, so it became possible to give it a new direction command while it was between blocks. For example, if it had just passed a block that said it could go right, pushing right from the neutral position would command it to go right even though there was a wall there. In short, I forgot to close gates when I was in the auto mode, because I assummed they were set by one of the four non-neutral positions. This is a fine example of misguided thinking.

In closing, we hope that we have provided you with enough programming techniques and game theory to create your own arcade games. Remember that originality, persistence and attention to detail are the keys to success in this industry. We hope some of our readers will join the ranks of successful Atari game designers. If you take the easy way out and program a quick game, the results will show in mediocracy.

APPENDIX

A: Useful PEEKs and POKEs

MEMORY CONFIGURATION

10,11 ($A,$B) Start Vector-Disk Based Software-(DOSVEC)
These locations hold the start vector or run the address for disk based binary load software. This is also the address BASIC jumps to when you call up DOS.

12,13 ($C,$D) Initialization Address for Disk Boot (DOSINI)
These locations hold the initialization address for the disk handler.

14,15 ($E,$F) Upper Limit of BASIC Program (APPMHI)
These locations contain the memory high limit for your BASIC program. Memory above that is used for screen display.

88,89 ($58,$59) Screen Memory Address (SAVMSC)
These addresses contain the lowest address of screen memory.

106 ($6A) Top of RAM Address-most significant byte (RAMTOP)
Gives the total number of pages (256 bytes) available. Peek (106)/4 gives the number of 1K blocks available.

741,742 ($2E5,$2E6) Free Memory High Address (MEMTOP)
This address is the highest free location in RAM for program and data. This value is updated when you press RESET, when you change GRAPHICS mode, or when a channel (IOCB) is OPENed to the display. The display list starts at the next byte above MEMTOP.

743,744 ($2E7,$2E8) Free Memory Low Address (MEMLO)
This is the address of the first free location of RAM for program use. This value which is normally 1792 ($700) is updated when DOS is present.

DISPLAY SCREEN

77 ($4D) Attract Mode On/Off (ATRACT)
Setting this location to 0 disables the attract mode, This happens automatically whenever a key is pressed. Normally this location is increnented every 4 seconds until the value reaches 127 ($7F), then set to 254 ($FE) until the attract mode is terminated. The attract mode protects the television screen by rotating the colors when the Atari is idle.

87 ($57) Display Mode (DINDEX)
This location contains the current BASIC graphics mode (0-11) in non-XL machines and (0-15) in XL machines. This can be used to fool the OS into thinking it is in a different graphics mode.

90 ($5A) Starting Graphics Cursor Row (OLDROW)
Used to determine the starting row for DRAWTO and XIO 18 (FILL) command.

91,92 ($5B,$5C) Starting Graphics Cursor Column (OLDCOL)
These locations are used by DRAWTO and XIO 18 (FILL) commands to determine the starting column of the DRAW or FILL.

96 ($60) Ending Graphics Cursor Row (NEWROW)
Row to which DRAWTO and FILL will go.

97,98 ($61,$62) Ending Graphics Cursor Column (NEWCOL)
Column to which DRAWTO and FILL will go.

660,661 ($294,$295) Split-screen Text Memory Address (TXTMSC)
Address of upper left corner of the split-screen text window.

708-712 ($2C4-$2C8) Playfield Color Registers (COLOR0-COLOR4)
Each of these locations determines the color for the various playfields. You can change these registers from BASIC via either a POKE or the SETCOLOR command.

752 ($2F0) Cursor Inhibit (CRSINH)
Zero turns the cursor on. Any other number turns the cursor off.

54276 ($D404) Horizontal Fine Scroll Register (HSCROL)
When horizontal fine scroll is enabled by setting bit 4 in the LMS instruction in the display list, POKEing this location with a value from zero to 16 clock cycles (depending on graphics mode) will fine scroll the screen.

54277 ($D405) Vertical Fine Scroll Register (VSCROL)
POKEing a value from zero to 16 (depending on graphics mode) will fine scroll the screen one or more scan lines. Vertical fine scrolling can only be used if bit 5 is set in the LMS instruction in the display list.

54282 ($D40A) Wait for Horizontal Synchronization (WSYNC)
This location allows the OS to synchronize the VBI's or DLI's with the screen display. Simply accessing this location halts the CPU until horizontal sync occurs.

54283 ($D40B) Vertical Line Counter (VCOUNT)
This register keeps track of the current line number currently be drawn on the screen. PEEKing here returns the line count divided by two; ranging from zero to 130 ($82). It is used during Display List Interrupts to change colors or when working with kernels.

CHARACTER SETS

755 ($2F3) Character Mode Register (CHACT)
This location defaults to a value of 2. Zero means normal inverse characters, one is blank (invisible) inverse characters, three is solid inverse characters. Four to seven is the same as zero to three, except the display is printed upside down. For example; POKE 755,4 will invert the standard character set.

756 ($2F4) Character Base Register (CHBAS)
This high byte location is used to tell ANTIC where the character set is located. It normally defaults to 224 ($E0) for upper case characters and numbers. Lower case

and graphics characters can be selected in graphics modes 1 and 2 by POKEing CHBAS with 226 ($E2).

763 ($2FB) Last ATASCII Character or Plot Point (ATACHR)

Returns the last ATASCII character read or written, or the value of a graphics point. The FILL and DRAW commands use this location for the color of the line drawn.

DISPLAY LISTS

512,513 ($200,$201) Display List Interrupt Vector (VDSLST)

These locations store the address of the instructions to be executed during a display list interrupt.

559 ($22F) DMA Control Register (SDMCTL)

This location enables or disables direct memory access by ANTIC. The normal default value is 34 ($22) which enables DMA for fetching normal playfield display data. You can turn the display off by POKEing a zero here.

560,561 ($230,$231) Display List Address (SDLSTL)

This is the starting address of the display list.

54286 ($D40E) Non-maskable Interrupt Enable (NMIEN)

This location is used to enable or disable vertical blank and display list interrupts. It is set to 64 ($40) upon powerup to enable VBI's. A value of 128 ($80) enables DLI's and 192 ($C0) enables both. A zero disables both.

KEYBOARD I/0

16 ($10) POKEY Interrupts (POKMSK)

This location enables and disables POKEY functions such as its timers, serial input/output data ready, and the BREAK key interrupt. The interrupt key can be disabled by POKEing a 112 to this shadowed location, and at 53774 ($D20E). It is better for the user to write his own routine for the BREAK key is reenabled whenever RESET is pressed or when PRINT or OPEN statements address the screen. You can store the location of your own interrupt routine at locations 566, 567 ($236,$237).

17 ($11) BREAK Key Flag (BRKKEY)

A zero means the BREAK key is pressed; any other number means it isn't.

764 ($2FC) Keyboard Character (CH)

This returns the value of the last key pressed. This value is neither internal or ATASCII but "raw" keyboard matrix code. A 255 ($FF) POKEd here clears it.

53279 ($D01F) Console Keys (CONSOL)

Used to determime if one of the three yellow console buttons have been pressed. It is best to clear it first with the value of eight to ensure an accurate reading. A value of 6 indicates the START key has been pressed, a 5 the SELECT key, and a 3 the OPTION key. The location reads a 7 when no CONSOLE keys are pressed, and a 0 when all are pressed simultaneously.

JOYSTICK/PADDLE I/O

624-631 ($270-$277) Paddle Game Controller (PADDL0-PADDL7)

A number between zero and 228 ($E4) is returned in these locations for each of the seven paddle potentiometers.

632-635 ($278-$27B) Joystick Controller Port (STICK0-4)

These locations return one of nine possible joystick positions for each stick. These values range from 5-15 ($5-$F). These values are 15 when the joystick is centered or in the neutral position.

636-643 ($27C-$283) Paddle Trigger (PTRIG0-PTRIG7)

Used to determine if the button on each paddle is pressed. A zero is returned if it is pressed and a one is returned when it isn't.

644-647 ($284-$287) Joystick Trigger (STRIG0-STRIG3)

These locations are used to indicate whether the joystick button is pressed. A zero is returned if it is pressed, and a one is returned when it isn't.

PLAYER-MISSILE GRAPHICS

623 ($26F) Player/Playfield Priorities (GPRIOR)

Priority options select which screen objects will be "in front" of the others. Players have presendence over all playfields when this location is one. A value of 4 gives the playfields priority over all of the players, and values 2 and 8 allow only some of the players to have priority over some of the players. In addition, the value 16 ($10) allows the four missiles to be combined into a fifth player, and a value of 32 ($20) allows players to be overlapped to produce a third color. This location is also used to enable the three GTIA modes.

704-707 ($2C0-$2C3) Player-missile Color Registers (COLPM0-COLPM3)

Each of these locations determines the color of a player and its corresponding missile.

53248-53251 ($D000-$D003) Player Horizontal Position Registers (HPOSP0-HPOSP3)

These write only registers determine the horizontal positions of each of the four players. Values range from 0- 277 ($0- $116).

53248-53251 (D000-$ D003) Missile to Playfield Collision Registers (MOPF-M3PF)

These read only registers tell you which playfield the missiles have collided with. The values are as follows; Playfield #0- 1, Playfield #1 -2, Playfield #2 -4, and Playfield #3- 8.

53252-53255 ($D004-$D007) Player to Playfield Collision Registers (POPF-P3PF)

These read only registers tell you which playfield the players have collided with. The values are the same as with the missiles above.

53252-53255 ($D004-$D007) Missile Horizontal Position Registers (HPOSM0-HPOSM3)

These write only locations determine the horizontal positions of each of the missiles. Values range from 0- 277 ($0- $116)

53256-53259 ($D008-$D00B) Player Width Registers (SIZE0-SIZE3)

These write only locations control the widths of the players. Values 0 and 2 produce normal width players, 1 double width players, and 3 quadruple width players.

53256-53259 ($D008-$D00B) Missile to Player Collision Registers (M0PL-M3PL)

These read only registers determine collisions between missiles and players. The values are as follows; with player #0 -1, player #1 -2, player #2 -4, and with player #3 -8.

53260 ($D00C) Missile Width Register (SIZEM)

This write only location controls the magnification of all four missiles. While a value of 0 or 2 displays all of the missiles at normal width, 1 displays all of the missiles at double width, and 3 displays all of the missiles at quadruple width, missiles widths can be set individually. Each missile is controlled in two bit pairs starting with the lowest. The bit values for the 0th missile is the same as above.

53260-53263 ($D00C-$D00F) Player to Player Collisions (P0PL-P3PL)

These read only registers return non-zero values when players collide. In general, the registers contain a 1 after a collision with player #0, a 2 after one with player #1, a 4 after one with player #2, and as 8 after one with player #3.

53277 ($D01D) Graphics Control Register (GRACTL)

This, when used in conjunction with DMACTL ($22F) enables player-missile graphics. A value of 2 enables player DMA only, a value of 1 enables missile DMA only, and a value of 3 enables both.

53278 ($D01E) Clear Player-missile Collision Registers (HITCLR)

You POKE this location with any number to clear the collision registers. This is normally done at the end of VBLANK after you have tested for all collisions.

54279 ($D407) Player-missile Base Registers (PMBASE)

This location contains the high byte starting address of your player-missile area.

SOUND

65 ($41) Input/Output Noise Control (SOUNDR)

Disk read/write operations can be silenced by POKEing a zero to this location.

53760,53762,53764,53766 ($D200,$D202,$D204,$D206) Audio Channel Frequency (AUDF1-3)

These locations control the frequency of the sound channels. The value N is used in the divide by N circuit. Thus the notes or tone becomes lower when N becomes larger. N can be in the range 1-256.

53761,53763,53765,53767 ($D201,$D203,$D205,$D207) Audio Channel Control (AUDC1-4)

These registers set the volume and distortion levels.

53768 ($D208) Audio Control Register (AUDCTL)

This location allows two channels to be joined for greater frequency range, and allows the user to vary the poly-counters which control noise, or to switch the clock base from 64KHz to 15KHz for a change in frequency range. To properly initialize the POKEY sound capabilities POKE AUDCTL with zero and POKE 53775,3 (D02F). This is equivalent in BASIC of SOUND 0,0,0,0.

MISCELLANEOUS

18,19,20 ($12,$13,$14) Internal Realtime Clock (RTCLOK)

Locations 20 increments every VBLANK interrupt or 1/60 second until it reaches 255 ($FF). Location 19 is then incremented by one and 20 is reset to zero. Likewise, location 18 is incremented when 19 reaches 255. This happens every 18.2 minutes.

53770 ($D204) Random Number (RANDOM)

This location produces a random number from 0 -255 ($0 -$FF) when read.

58454 ($E456) Central Input/Output (CIO) Utility Entry (CIOV)

This is actually a subroutine in the OS ROM that is used to pass I/O operations to the correct device driver. Once parameters are set in the correct IOCB's you jump here to use them. While BASIC only supports one byte-at-a-time IO (GET and PUT), addressing CIOV directly allows the user to input or output a buffer of characters at a time.

58460 ($E45C) Set Vertical Blank (SETVBK)

This OS routine, which sets system timers during the VBLANK routine, insures that both bytes of the vector addressed will be updated while VBLANK is enabled.

58466 ($E462) Exit Vertical Blank (XITVBK)

This OS subroutine is used to restore the computer to its pre-interrupt state upon exiting the user's VBLANK routine. The computer can then resume normal processing.

B: ATASCII Character Set

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER
0	0	
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	
10	A	
11	B	
12	C	
13	D	
14	E	
15	F	
16	10	
17	11	
18	12	
19	13	
20	14	
21	15	
22	16	
23	17	
24	18	
25	19	
26	1A	
27	1B	
28	1C	
29	1D	
30	1E	
31	1F	
32	20	Space
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&

APPENDIX

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER
39	27	'	55	37	7	71	47	G
40	28	(	56	38	8	72	48	H
41	29	)	57	39	9	73	49	I
42	2A	*	58	3A	:	74	4A	J
43	2B	+	59	3B	;	75	4B	K
44	2C	,	60	3C	<	76	4C	L
45	2D	-	61	3D	=	77	4D	M
46	2E	.	62	3E	>	78	4E	N
47	2F	/	63	3F	?	79	4F	O
48	30	0	64	40	@	80	50	P
49	31	1	65	41	A	81	51	Q
50	32	2	66	42	B	82	52	R
51	33	3	67	43	C	83	53	S
52	34	4	68	44	D	84	54	T
53	35	5	69	45	E	85	55	U
54	36	6	70	46	F	86	56	V

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	
124	7C	|
125	7D	
126	7E	
127	7F	

C: Assembler Comparisons

SYN ASSEMBLER F-S MACRO	ATARI ASSEMBLER	MAC 65	ATARI MACRO	
.OR $600	* = $600	.OR $600 or * = $600	ORG $600	
.EQ	=	.EQ or =	EQU or =	
.BS 5	* = * + 5	.DS 5	DS 5	
.HS FFFFFF	.BYTE $FF,$FF,$FF	.BYTE $FF,$FF,$FF	DB $FF,$FF,$FF	
.DA #20,#40	.BYTE 20,40	.BYTE 20,40	DB 20,40	
.DA $E474 .DA START	.WORD $E474 .WORD START	.WORD $E474 .WORD START	DW $E474 DW START	
.AT "HELLO" (NOTE)	.BYTE Using internal HEX values	.SBYTE "HELLO"	DB Using internal HEX values	
.AS "HELLO"	.BYTE "HELLO"	.BYTE "HELLO"	DB "HELLO"	
# LABEL	# LABEL & $FF	#< LABEL	# LOW LABEL	
/ LABEL	# LABEL / 256	#> LABEL	# HIGH LABEL	
BGE LABEL	BCS LABEL	BCS LABEL	BCS LABEL	
BLT LABEL	BCC LABEL	BCC LABEL	BCC LABEL	
.IN "D:PARTZ"	———	.INCLUDE "#D:PARTZ"	INCLUDE D:PARTZ	

NOTE: The F-S Macro Assembler uses a .AS ^ "HELLO" where the ^ is a Shift * sign.

	EASTERN HOUSE	MEANING
	.BA $600	Define program origin
	.DE	Define equates
	.DS 5	Reserves space for data
	.BY $FF,$FF,$FF	Define Hexidecimal data
	.BY 20 40	Defines bytes
	.SI $E474 .SI START	Define a two byte word low byte, high byte order
	.BY Using internal HEX values	Define string using internal character values
	.BY "HELLO"	Define string using ASCII values
	#L, LABEL	Returns Low byte (LDA # LABEL)
	#H, LABEL	Returns High byte (LDA / LABEL)
	BCS LABEL	Branch if > = (After a compare)
	BCC LABEL	Branch if < (After a compare)
	.FI "DI:PARTZ"	Include a file for Assembly

D: Binary File Autorun

Since assembly language files (object code) generated by Syn-Assembler and many other assemblers will not run automatically from the DOS menu without specifying the run address, we have provided a BASIC language utility that will fix the problem. Files run automatically from Atari DOS if the starting address in low byte, high byte order is appended to the file, DOS reads these two values into locations $2E0 and $2E1 (736, 737). Upon completion of the binary load, control is normally passed back to the DOS menu. However, if there is an address in these locations, the computer will jump to that location.

Our utility will append your file automatically, a function that used to be provided with the /A command in the old DOS 1.0 menu. The program only asks for the name of the file and its run address in decimal. Assuming the file in on the disk in the disk drive, the utility will open the file, append the run address, then close it. It will now run automatically when you load it from the DOS menu using the L command.

```
10 REM MAKES BINARY FILE AUTORUN -DAN PINAL
20 REM THE DOS RUN ADDRESS IS AT $2E0 & $2E1
30 REM THIS ROUTINE APPENDS A NEW BINARY LOAD HEADER TO LOAD INTO $2E0 TILL $2E1
40 REM THEN PUTS THE LOW & HIGH TO THE FILE AND CLOSES IT
50 DIM Q$(14),F$(14):F$="D:"
60 PRINT "NAME OF FILE TO APPEND";
70 INPUT Q$:F$(3)=Q$
80 OPEN #1,9,0,F$
90 REM PUT IN NEW HEADER $2E0 - $2E1
100 PUT #1,224:REM $E0
110 PUT #1,2:REM $02
120 PUT #1,225:REM $E1
130 PUT #1,2:REM $02
140 PRINT "INPUT RUN ADDRESS IN DECIMAL";
150 INPUT A
160 HI=INT(A/256):LO=A-HI*256
170 PUT #1,LO
180 PUT #1,HI
190 CLOSE #1
```

E: Source Code for Chapters 3 & 5

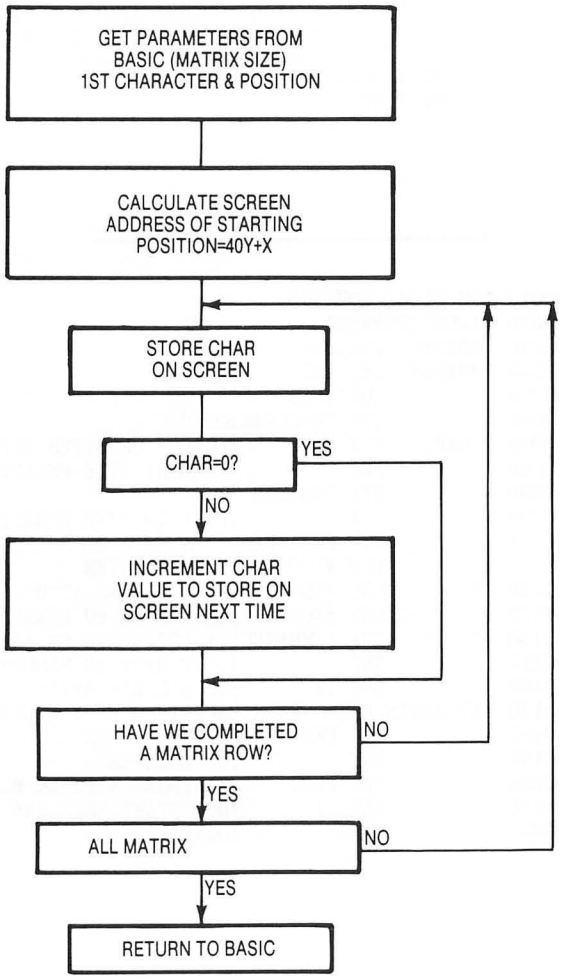
```
                00010 *CHARACTER SET MOVE ROUTINE
                00020 *A=USR(1664,CHRAMH,1 OR 2,PAGES)
                00030 *
                00040          .OR $680
                00050          .TF "D:CHARMOVE.OBJ
00CB:           00060 CHRAML   .EQ $CB
00CC:           00070 CHRAMH   .EQ $CC
00CD:           00080 CHROML   .EQ $CD
00CE:           00090 CHROMH   .EQ $CE
0680: 68        00100 START    PLA          ;PULL UNUSED BYTE OFF STACK
0681: 68        00110          PLA          ;DISCARD LO BYTE
0682: 68        00120          PLA          ;RAM LOCATION OF CHARACTER SET
0683: 85 CC     00130          STA CHRAMH
0685: 68        00140          PLA
0686: 68        00150          PLA          ;START AT 1ST OR 2ND HALF?
0687: C9 01     00160          CMP #$01
0689: F0 04     00170          BEQ .1       ;IF 2 SKIP 2 PAGES
068B: E6 CE     00180          INC CHROMH
068D: E6 CC     00190          INC CHRAMH
068F: 68        00200 .1       PLA
0690: 68        00210          PLA          ;# PAGES
0691: 8D AC 06  00220          STA PAGES
0694: A9 00     00230          LDA #$00
0696: 85 CE     00240          STA CHROMH
0698: 85 CC     00250          STA CHRAMH
069A: A8        00260          TAY
069B: B1 CD     00270 LOOP     LDA (CHROML),Y ;LOAD FROM ROM
069D: 91 CB     00280          STA (CHRAML),Y ;STORE IN RAM
069F: C8        00290          INY
06A0: D0 F9     00300          BNE LOOP     ;DONE WITH 256 BYTE PAGE?
06A2: E6 CE     00310          INC CHROMH   ;NEXT ROM PAGE
06A4: E6 CC     00320          INC CHRAMH   ;NEXT RAM PAGE
06A6: CE AC 06  00330          DEC PAGES
06A9: D0 F0     00340          BNE LOOP     ;DONE?
06AB: 60        00350          RTS
06AC:           00360 PAGES    .BS 1
```

```
        00010 *SOURCE CODE FOR LOAD OR SAVE CHARACTER SETS - DAN PINAL
        00020 * CIO CALL TO SAVE OR LOAD CHRS
        00030 * IOCB MUST BE OPPENED
        00040 CIO
68      00050          PLA          ; GET # ARGS OFF STACK
68      00060          PLA          ; HIBYTE OF IOCB TO USE
68      00070          PLA          ; IOCB
0A      00080          ASL
0A      00090          ASL
0A      00100          ASL
0A      00110          ASL          ; X16
AA      00120          TAX
68      00130          PLA          ; BUFFER HI
9D 45 03 00140         STA $345,X
68      00150          PLA          ; BUFFER LO
9D 44 03 00160         STA $344,X
68      00170          PLA          ; LENGTH HI
9D 49 03 00180         STA $349,X
68      00190          PLA          ; LENGTH LO
```

```
9D 48 03 00200          STA $348,X
68       00210          PLA          ; Q$ HI
85 D5    00220          STA $D5
68       00230          PLA          ; Q$ LO
85 D4    00240          STA $D4
A0 00    00250          LDY #$00
B1 D4    00260          LDA ($D4),Y
A0 07    00270          LDY #$07     ; FOR LOAD
C9 53    00280          CMP #'S      ; SAVE/LOAD?
D0 02    00290          BNE .1
A0 0B    00300          LDY #$0B     ; THEN SAVE
         00310 .1
98       00320          TYA
9D 42 03 00330          STA $342,X   ; COMMAND
4C 56 E4 00340          JMP $E456    ; CIOV
```

```
         00010 ; MATRIX CYCLER
         00020 ; DAN PINAL
         00030 ;         .TF "D:MCYCLER.OBJ"
         00040 ;         .LI OFF
         00050 ; CALL FROM BASIC
         00060 ; X=USR(MCYCLER,ROW,COL,MATRIXH,MATRIXV,STARTING CHARACTER)
         00070 ; IF STARTING CHARACTER=0 THEN WILL ERASE MATRIX
         00080 ;
         00090 POTMPO    .EQ $D4
         00100 POTMP1    .EQ $D5
         00110 HLIMIT    .EQ $CE
         00120 VLIMIT    .EQ $CF
         00130 SCREENLO  .EQ $58
         00140 SCREENHI  .EQ $59
         00150 ;
         00160 MCYCLER
68       00170          PLA          ; GET # OF ARGS OFF STACK
68       00180          PLA          ; HI BYTE OF ROW - SHOULD BE 0
68       00190          PLA          ; ROW
A8       00200          TAY
A6 59    00210          LDX SCREENHI
68       00220          PLA
68       00230          PLA          ; COLUMN
18       00240          CLC
65 58    00250          ADC SCREENLO
90 01    00260          BCC .1
E8       00270          INX
         00280 .1
88       00290          DEY
30 08    00300          BMI .2       ; ONLY EXIT
18       00310          CLC
69 28    00320          ADC #$28     ; +40 FOR NEXT ROW
90 F8    00330          BCC .1
E8       00340          INX
D0 F5    00350          BNE .1       ; ALWAYS
         00360 .2
85 D4    00370          STA POTMPO
86 D5    00380          STX POTMP1
68       00390          PLA
68       00400          PLA
85 CE    00410          STA HLIMIT
68       00420          PLA
68       00430          PLA
85 CF    00440          STA VLIMIT
68       00450          PLA
```

MATRIX CYCLER




```
68          00460           PLA
AA          00470           TAX
            00480 PLOT
A0 00       00490           LDY #$00
            00500 .1
91 D4       00510           STA (POTMPO),Y
C9 00       00520           CMP #$00    ; ERASING?
F0 02       00530           BEQ .2
E8          00540           INX
8A          00550           TXA
            00560 .2
C8          00570           INY
C4 CE       00580           CPY HLIMIT
D0 F3       00590           BNE .1
18          00600           CLC
```

```
A9 28     00610          LDA #$28    ; +40 FOR NEXT ROW
65 D4     00620          ADC POTMPO
85 D4     00630          STA POTMPO
90 02     00640          BCC .3
E6 D5     00650          INC POTMP1
          00660 .3
8A        00670          TXA
C6 CF     00680          DEC VLIMIT
D0 E1     00690          BNE PLOT
          00700 ;
60        00710          RTS
          00720 ;
```

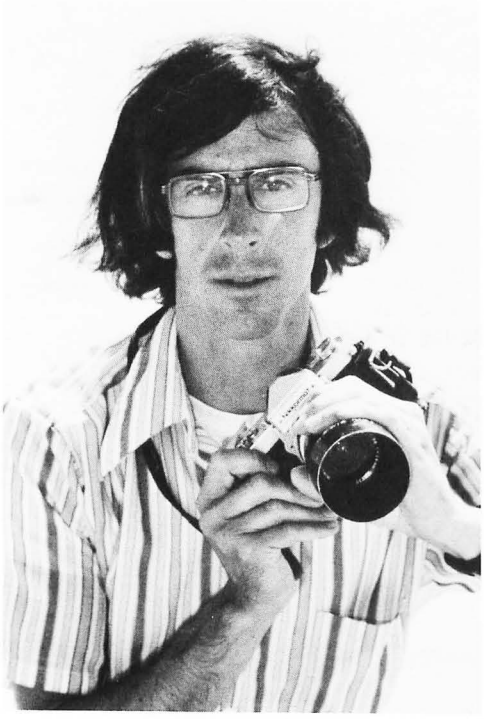
```
                00010 *PM CLEAR ROUTINE
                00020 *A=USR(PMBASE)
00D4:           00030 PMBASEL  .EQ $D4
00D5:           00040 PMBASEH  .EQ $D5
                00050          .OR $6A0
                00060          .TF "D:CLEAR.OBJ"
06A0: 68        00070 CLEAR    PLA          ;PULL # OF BYTES OFF STACK
06A1: 68        00080          PLA          ;PULL HI BYTE PMBASE
06A2: 85 D5     00090          STA PMBASEH
06A4: 68        00100          PLA          ;PULL LO BYTE PMBASE
06A5: 85 D4     00110          STA PMBASEL
06A7: A2 00     00115          LDX #$00     ;PAGE COUNTER
06A9: A0 00     00120 .1       LDY #$00     ;PAGE BEGINS AT 0
06AB: A9 00     00130          LDA #$00     ;LOAD ZERO TO ERASE
06AD: 91 D4     00140 .2       STA (PMBASEL),Y ;STORE IN PM AREA
06AF: C8        00150          INY          ;NEXT BYTE IN MEMORY
06B0: D0 FB     00160          BNE .2       ;DO ALL 256 BYTES
                00170 *AT 256TH BYTE WRAPS BACK TO 0 IN Y REGISTER; FALLS THROUGH
06B2: E6 D5     00180          INC PMBASEH ;DO NEXT PAGE
06B4: E8        00190          INX          ;UP COUNTER
06B5: E0 08     00200          CPX #$08     ;FINISHED WITH 2K BLOCK
06B7: 90 F0     00210          BLT .1       ;NO, START NEXT 256 BYTE PAGE
06B9: 60        00220          RTS          ;DONE!
```

INDEX

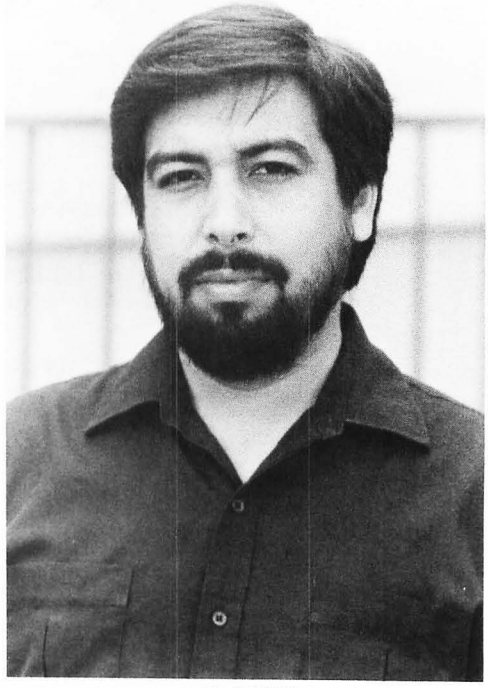
ABOUT THE AUTHORS

Jeffrey Stanton received a BME (1967) and a MSME (1969) from Rensselaer Polytechnic Institute. He worked as a control systems engineer and mechanical engineer for the aerospace industry in the early 1970's. His interest in computer game design sidetracked his career as a photographer and book illustrator in the late 1970's. In addition to writing several Apple arcade games and doing some occasional consulting, he is the author of *Apple Graphics and Arcade Game Design*, and one of the editor/reviewers for the books of *Apple and Atari Computer Software*. He currently divides his time between writing and reviewing software in the mornings, and operating a postcard stand on the Venice Beach boardwalk in the afternoons. He lives in Venice, California.

Dan Pinal, typical of many of the early computer hobbyists, is self educated. He was one of the first to own an Atari computer, and entered the micro-computer industry a year later. Dan consulted, taught and did game programming for two software houses at the peak of the game market in 1983. He has one Atari game currently on the market. Dan currently lives in Los Angeles, California.



JEFFREY STANTON



DAN PINAL

PROGRAM LISTINGS & ASSEMBLER

The code listed in this book, available on diskette only, and the F-S Macro Assembler 40/80 can be ordered using this coupon.

Please send me:

( ) BASIC program listings only: **$10.00** each _____

( ) Assembly language listings only: **$15.00** each _____

( ) All program listings (2 disks - includes playable games): **$20.00** each _____

( ) Playable games only: **$12.50** each _____

( ) F-S Macro Assembler 40/80: **$50.00** each _____

Sales tax (California residents add 6½%) _____

Postage and handling (source listings — $1.50) _____

Postage and handling (F-S Macro Assembler — $1.50) _____

TOTAL _____

[ ] Enclosed is my check or money order (Sorry, no credit cards), payable to STANTON PRODUCTS.

Name _____

Address _____

City _____ State _____ Zip _____

Mail to: STANTON PRODUCTS
3710 PACIFIC AVENUE #16
MARINA DEL REY, CALIFORNIA 90292

Other ATARI Books Available from Arrays, Inc./The Book Division

( ) The Book of Atari Software - 1985: **$19.95** _____

( ) The Atari User's Encyclopedia: **$19.95** _____

( ) Atari Color Graphics: A Beginner's Workbook: **$12.95** _____

Sales tax (California residents add 6½%) _____

(Postage will be paid by Arrays, Inc./The Book Division)

TOTAL _____

[ ] Enclosed is my ☐ check, ☐ money order, ☐ Visa or ☐ MasterCard

_____ Signature _____

Payable to Arrays, Inc.

Name _____

Address _____

City _____ State _____ Zip _____

Mail to: ARRAYS, INC./THE BOOK DIVISION
11223 SOUTH HINDRY AVE.
LOS ANGELES, CALIFORNIA 90045

ATARI GRAPHICS and ARCADE GAME DESIGN

Atari Graphics and Arcade Game Design *is the most comprehensive book about designing arcade games on the market. More than just a book on designing games, it teaches all of the basic fundamentals of Atari graphics including scrolling, GTIA color, display lists, player-missile graphics, character animation, vertical blank and display list interrupts.*

The book is understandable and informative to programmers on all levels ranging from intermediate BASIC to advanced Assembly language. ***Atari Graphics and Arcade Game Design*** *addresses an audience of teenagers and young adults, so it presents graphics concepts simply through clear text, diagrams, and easy to follow examples. The book thoroughly flowcharts and explains the sometimes complex logic of the game code discussed. Subjects such as bomb drops, laser fire, missile and ship movement are covered in great detail as are the physics and mathematical concepts needed to make them work realistically. And useful tools such as character and player-missile editors are included as an added bonus.*

Atari Graphics and Arcade Game Design *teaches game design using advanced Atari graphics principles. As examples, it develops a wide variety of complete working games in both BASIC and Assembly language. The first two example games are first written in BASIC, then translated into Assembly language as a learning bridge between BASIC and the faster, more powerful Assembly language. A third BASIC game demonstrates that you can create playable moderate-speed games in BASIC with the help of machine language routines developed in this book. Finally, three commercial quality machine language arcade games—an alphabet maze, a scrolling game with ground laser bases and a programmable alien attack, and a two-player tank game with rotating turrets and blow-away walls—help advanced programmers pursue state-of-the-art game design.*

ISBN 0-912003-05-7

$16.95 (U.S.A.)

ARRAYS, INC.
THE BOOK DIVISION