BASIC TURBOCHARGER

Machine Language Routines for BASIC programs

By Jeff Bader

ALPHA SYSTEMS

BASIC TURBOCHARGER

By Jeff Bader

ALPHA SYSTEMS

Atari, Atari 800 Computer, Atari 800XL Computer, Atari 130XE Computer, Atari 1200XL Computer are all trademarks of Atari Corp.

1st Printing July, 1987

Manufactured in the United States of America

10 9 8 7 6 5 4 3

Acknowledgements

To those who made this book possible; Atari Corp, Alpha Systems, and my parents.

BASIC TURBOCHARGER
TABLE OF CONTENTS

INTRODUCTION

Welcome to the exciting world of Atari machine language. Machine language is the fastest and most memory efficient language available. Many of the powerful features of the Atari 8-bit computers can only be accessed through the use of machine language.

But BASIC programmers have no fear! You don't need to learn any machine language or Assembly Language (Assembly is the language used to write machine language) to use the programs in this book. All the more than 160 machine language routines presented in this book are designed to be easily included and executed from a BASIC program. Each routine is included in a BASIC demonstration program that is fully documented in the book and included on the disk. No typing required!

Now you can load and save picture screens from the popular picture creation programs on the market including Micro-Painter and compressed format Koala and Atari Touch Tablet pictures. You can do scrolling, array sorting, hex, binary, and decimal numerical conversions, joystick reading, Player/Missiles, multi-color screens, string searches, picture printing, and many more operations using machine language.

To the beginning or even experienced Assembly Language programmer this book will prove invaluable. With the Assembly Source Code also available on disk by separate purchase, the machine routines in this book will provide a solid and varied foundation of working programs to build upon.

But now no matter what level of programmer you are you can write 'turbocharged' programs! So let's get started!

Chapter 1
A Little About BASIC
and
The Routines in this book

Reading the following material will be of great help in understanding how to properly execute and use the machine routines in this book in your own BASIC programs. If you have any problems, the answers will most likely be found in this section.

Although no programming skills are required to run the sample BASIC programs you'll need at least a beginners level of BASIC programming knowledge to understand and use the machine routines in your own programs. We recommend your first steps into BASIC programming come from Atari's 'Owners Manual' and your next steps from the book 'Your Atari Computer' by Osborne/McGraw-Hill.

A LITTLE ABOUT BASIC

The 'USR' Statement

Machine language code is a series of numbers stored in the computer memory. Each number stands for an instruction that the 6502 central processing unit of the Atari 8-bit computer can understand. The BASIC 'USR' command will run the machine code from a BASIC program. All we need to know is where in the computer memory the machine language code resides.

The USR call has the general form:

```
X=USR(Address, P1, P2, ...)
```

Where:

Address	The decimal address of the machine code in the computer memory.
P1,P2,...	Numbers, called parameters, that are to be passed from the BASIC program to the machine program. None or several parameters may be required by the machine program. A parameter must be a positive decimal integer from 0 to 65,535.
X	The Return Variable. Any legal Atari variable name may be used. The Return Variable will always contain a number from 0 to 65,535 after the machine code has been executed. This number is meaningless unless the machine program was designed to return a useful number back to the BASIC program.

The USR statement says to the BASIC program: "Go to the 'Address' indicated and you will find the beginning of a machine language program. Execute the program and use the numbers P1,P2,... when requested by the machine program. When the machine routine is completed go back to the BASIC program, assign a value (0-65535) to the variable 'X' as instructed by the machine program, and continue processing."

The parameters passed to the machine program can be any legal variable name, number, or equation. For example, the following two USR calls are identical.

```
10 A=12:D=5
20 B=3*A+D
30 X=USR(Address,A,B)
```

or

```
10 A=12:D=5
20 X=USR(Address,12,3*A+D)
```

The location (Address) of the machine code in the computer memory must be known either directly or indirectly.

Three methods are commonly used.

1) Direct Memory Access.

In this method the machine code is directly placed at a known and unchangeable location in memory. The BASIC program will load the machine code from either the disk, DATA statements, or a string and place it into the specified memory.

LOAD the file 'DEMO' from the front of the program disk and study the first program, DEMO1, in lines 10 to 70. The actual machine code is contained in the DATA statements in lines 40 through 60. Line 10 reads the machine code from the DATA statements and places (POKE's) them into the computer memory starting at memory location 1536, the start of Page 6. Line 10 also clears the screen. Page 6 is the one of the most popular places to send machine code. This area of 256 bytes of memory was set aside by the by the designers of the Atari computers for use by the programmer. It is, normally, not used by the computer and is a safe place to put data, machine programs, tables, etc. This area of memory starts at decimal address 1536 and extends to 1791.

Line 20 is the USR call that runs the machine program which places the Alphabet on the screen. The 'Address' specified in this USR call is 1536 where the machine program was just placed. Two parameters are required. The first parameter is the line number (0-23 for a GRAPHICS 0 screen) on which to locate the Alphabet. The second parameter moves the string horizontally on the screen. RUN the program and try different values for the two parameters.

```
0 REM DEMO.1
10 FOR I=0 TO 57:READ A:POKE 1536+I,A:
NEXT I:? CHR$(125)
20 X=USR(1536,9,7)
30 END
40 DATA 104,165,89,133,204,165,88,133,
203,104,104,170,240,14,165,203,24,105,
40,144,2,230,204,202,208
50 DATA 246,133,203,24,104,104,101,203
,144,2,230,204,133,203,162,26,160,0,16
9,33,133,205,165,205,145
60 DATA 203,230,205,200,202,208,246,96
70 END
```

Many of you have, no doubt, run into this kind of machine language programming when typing in programs from magazines. All those numbers in the DATA statements are the actual machine language code. Using DATA statements is very popular because it is a fairly safe way to type in a program from a listing without making mistakes. However, READing all those DATA numbers into memory in your BASIC program does take time and is memory wasteful. The next two methods of accessing machine code speeds up the process and saves memory space.

2) String Addressing

To speed up the process of getting to our machine code the numbers in the DATA statements can be placed into an ATARI string and the address of the string used in the USR statement. This is exactly what the second program, DEMO2, in the file 'DEMO' does. The string P$ contains the character representation for all those numbers (ATASCII) in the DATA statements from the previous program. This conversion process can be accomplished by using the tables in your Atari's 'OWNER'S MANUAL', the table in the Appendix, or by using the following BASIC statement.

```
? CHR$(NUMBER)
```

There are some problems encountered when displaying string characters on the screen. First, some characters (27-31, 123-127, 187-191, and 251-255) can not be printed to the screen because they perform screen editing functions. To allow them to be printed to the screen POKE 766,1 (Default = 0) which deactivates their screen editing functions.

The second problem concerns the characters for numbers 34 (Quotation mark) and 155 (End-of-Line). The quotation mark can not be placed inside a string because it would signal the end of the string. The End-of-Line character has no character. However, these numbers can be placed indirectly into the string by use of the CHR$ command. For example, the string is first set up with any character temporarily holding the place of these two characters wherever encountered. Once completed, the two problem characters are placed into the string indirectly by use of the BASIC 'CHR$' command.

P$(X,X)=CHR$(34) or
P$(X,X)=CHR$(155)

X stands for the location within the string where these characters are to be placed. Programs number 54 and 55 use this technique. Now that the machine code is properly placed into a string the address of the string can be found by the BASIC 'ADR' command. The computer will find the location of the string and protect it from being stepped on by the rest of the BASIC program.

Now back to our DEMO2 program. Line 100 does the necessary DIMension of the string, P$, and clears the screen. Line 120 is the USR call to the address of the string, P$, and uses the same two parameters as the first program. Notice, the program executes immediately when RUN because there is no time required to fill a memory area with all those numbers in the DATA statements.

```
90 REM DEMO.2
100 DIM P$(58):? CHR$(125)
110 P$="h[illegible]Y[illegible]X[illegible]hh[illegible]i([illegible]h
he[illegible]!
[illegible]"
120 X=USR(ADR(P$),9,7)
130 END
```

3)Direct Access

The third and most memory efficient way to run a machine routine is to access the characters directly. If all the characters and the rest of the USR call can be placed on one logical line (approximately three screen lines) then the actual characters can be substituted for the address of the string (POKE 82,0 will give you more line space by moving the left margin from 2 to 0).

The third and last program on the file 'DEMO' shows this technique which again gives the same results as the other two programs.

```
150 REM DEMO.3
160 DIM P$(58):? CHR$(125)
170 X=USR(ADR("h[illegible]Y[illegible]X[illegible]hh[illegible]i([illegible]
[illegible]hhe[illegible]![illegible]"),
9,7)
```

Whenever possible, this book uses this method of calling a machine program. If you would like to convert all those funny looking characters to their ATASCII decimal value you can again use the table in the Appendix or use the table in Atari's 'Owners Manual'.

In order to use methods 2 & 3 the machine code must be written in what is called relocatable code. What is relocatable code? It simply means that the machine language routine can be placed anywhere in memory and it will function properly. Non-relocatable code must be loaded into an exact memory location regardless of what program or computer is being used. It is easier and more memory efficient to write a machine program in non-relocatable code but only method 1 can be used to access the code. All the machine language routines in this book were written using relocatable code.

Strings

A special precaution must be understood about the use of Atari strings which will save you countless debugging time. This precaution concerns the finding of the address of the string.

Every string must first be DIMensioned to the maximum length that string will see in your program. The location in the computer memory, however, is not set until the string has been assigned its first values. Consequently, if you try to find the address of the string before it's assigned, with the BASIC 'ADR' command, the wrong location may result. Therefore, if the string has not been assigned values at least once before you need it then use the following BASIC statement to set and clear the string.

```
P$=" ":P$(X)=P$:P$(2)=P$
```

In this example 'X' stands for the number in the DIM statement for the string, the maximum length of the string. This statement is used throughout this book. Notice that this line also clears the string. It can also be used to fill a string with any character of choice by replacing the blank between the quotation marks with the desired character.

PEEKing & POKEing

If you understand PEEKs and POKEs then skip to the next section. These two BASIC commands are used throughout this book and to the beginning BASIC programmer their use may be a mystery. In reality, they are really very simple commands.

A PEEK just means to look at the decimal number in one memory register. A POKE places a number in one memory register. These numbers can be any integer value from 0 to 255. When you encounter these commands in the programs in this book they will be explained.

Since a memory register can only hold a number up to 255 then how can the computer address its full memory potential of 65535 bytes? It does this by breaking up these higher numbers into two parts and placing them into consecutive memory locations. The Most Significant Byte (MSB) of the original number is the integer result of the number divided by 256. The Least Significant Byte (LSB) is the original number minus the MSB. The LSB is always placed into the first memory location and the MSB byte is placed in the following location. The computer knows to multiply the MSB by 256 and add the LSB to get the original number. You will run into these two byte registers often in this book when dealing with finding the address of the start of the screen memory and the screen's Display List.

THE ROUTINES IN THIS BOOK

To run the various sample BASIC programs in this book, first turn on your disk drive and insert side 1 of the program disk. Side 1 contains the first 56 programs, DOS, the DEMO program, and various sample character sets and pictures. Side 2 contains programs 57 through 120.

Turn on your computer, with BASIC installed, and wait for the familiar 'READY' prompt. Most of the programs on the disk are SAVEd to disk, requiring the LOAD or RUN command to operate. A few programs are LISTed to the disk (read their explanations) and require the ENTER command. All the programs have the general filename format of

PROGRAM.XXX

where XXX is the desired program number, 001 to 120.

There is no room on the disk to save your own programs. All filename space is used. Therefore, there is no need to write to our disk. The disk is not copy protected, so you should make a backup copy for safe storage. We request you respect our decision of not protecting the disk for your convenience by making copies only for your backup purposes. The large amount of work and expense that goes into producing a low cost high quality product for the Atari computers can only be justified by making a reasonable return.

Feel free to use any of these routines in your own programs whether for fun or for profit. You have our permission to use them in your own commercial programs, with out any license fees. We do ask, though,that you mention this book in your acknowledgments.

When you use these routines, be careful when renumbering them. They may not work if they are renumbered improperly.

Chapter 2
Moving Memory

Moving memory from one area to another within the computer is one of the best uses for machine language. Even very large blocks of memory can be moved thousands of times faster than in BASIC. You can use these routines for copying graphics screens to other areas of memory, placing string data into memory (and vice versa), and for all types of memory moving.

PROGRAM.001 Move Memory: Right

The following program contains two memory moving machine routines. The first one in line 30 moves up to a maximum of 256 (characters) bytes of memory. The second routine in line 60 moves any amount of memory. These routines move memory starting from the first byte to the last byte. The destination area may overlap the source area, except for the last location. The source data is not erased but copied to the destination area. The sample program below shows the moving of the data in string S$ to Page 6 starting at location 1536. Line 40 prints out the data in the new location to prove the move took place. Three parameters are required to be passed into the machine program.

P1 Starting source address.
P2 Starting destination address.
P3 Length of data to move.

```
0 REM PROGRAM.001
10 DIM S$(105)
20 S$="MOVE THIS STRING INTO PAGE SIX
STARTING AT ADDRESS 1536.  MOVING MEMO
RY BY MACHINE LANGUAGE IS VERY FAST."
30 X=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]hh[illegible]
[illegible]"),ADR(S$),1536,LEN(S$)):REM *
   MAXIMUM LENGTH IS 256.
40 FOR I=0 TO LEN(S$)-1:? CHR$(PEEK(15
36+I));:NEXT I
50 END
60 X=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]h[illegible]h[illegible]
[illegible]"),P1,P2,P3)
:REM USE TO MOVE ANY LENGTH.
```

PROGRAM.002 Move Memory: Left

It may be necessary to move the source memory to another area of memory (higher) that partially overlaps the source area. In this case we need to move the source memory starting from it's last location and work down to it's beginning. The machine code to accomplish this move is a little more involved and slower (Of course, slower in machine language is still almost to fast to measure!) than the PROGRAM.001 move routines. The memory areas can overlap all but the first location.

Line 10 places the alphabet into Page 6 memory starting at location 1536. The machine routine in line 20 (maximum move is 256) then shifts the data up in memory 10 bytes starting at 1546. Line 30 checks to make sure the move took place. The same parameters are used as in PROGRAM.001.

P1 Starting source address.
P2 Starting destination address.
P3 Length of data to move.

```
0 REM PROGRAM.002
10 FOR I=0 TO 25:POKE 1536+I,I+65:NEXT
 I
20 X=USR(ADR("hh[illegible]"),1536,1546,26):REM MAXIMUM LENGTH
   IS 256.
30 FOR I=0 TO 25:? CHR$(PEEK(1546+I));
:NEXT I
40 END
50 X=USR(ADR("hh[illegible]e[illegible]
e[illegible]h[illegible]
"),P1,P2,P3):REM USE FOR ANY LENGTH.
```

PROGRAM.003 Move Memory & Reverse

This program is like PROGRAM.001 except it sends the memory to the new location in reverse order. The sample program shows the reversing of memory between two strings. Line 30 shows the very important step of setting and establishing the length of the destination string, B$(see Introduction Section) since it is the first time this string has been encountered.

P1 Starting source address.
P2 Starting destination address.
P3 Length of data to move.

To send and reverse memory over 256 bytes in length use the machine routine in line 60 of PROGRAM.001 then use line 70 of PROGRAM.004.

```
0 REM PROGRAM.003
10 DIM A$(26),B$(26)
20 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30 B$=" ":B$(26)=B$:B$(2)=B$
40 X=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]hh[illegible]♥[illegible]
[illegible]◆"),ADR(A$),ADR(B$),
LEN(A$)):REM MAXIMUM LENGTH IS 256.
50 ? B$
```

PROGRAM.004 Reverse Memory

The following program reverses the order of the contents of the memory range specified. Line 30 contains the routine to reverse up to 256 bytes of memory and lines 70 and 80 contain the routines to reverse any length.

The sample program demonstrates the reversing of the alphabet in the string A$. Two parameters are required.

P1 Starting source address.
P2 Length of data to move.

```
0 REM PROGRAM.004
10 DIM A$(26)
20 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30 X=USR(ADR("hh[illegible]h[illegible]hh[illegible]j[illegible]♥[illegible]◆"),ADR(
A$),LEN(A$)):REM MAXIMUM LENGTH IS 256
40 ? A$
50 END
60 REM **USE THE FOLLOWING ROUTINE TO
   REVERSE MEMORY OF ANY LENGTH. **
70 P$="hh[illegible]h[illegible]h[illegible]h[illegible]e[illegible]e[illegible]i [illegible]
[illegible]i ♥[illegible]F[illegible]f[illegible]♥[illegible]
[illegible]"
80 X=USR(ADR(P$),P1,P2)
```

Chapter 3
Fun With Text

PROGRAM.005 Text: Upper Case To Lower Case

The following program will change UPPER case letters to lower case. Simply specify the starting address of the string or area of memory to change and the length of that memory to change. The routine in line 30 is for a maximum length of 256 and the routine in line 60 is for any length. Two parameters are required.

P1 Starting address of memory to change.
P2 Length.

```
0 REM PROGRAM.005
10 DIM A$(26)
20 A$="CHANGE UPPER TO LOWER CASE"
30 X=USR(ADR("hh[illegible]h[illegible]hh[illegible]A[illegible]i [illegible]"),ADR(A$),LEN(A$)):REM MAXI
MUM LENGTH IS 256.
40 ? A$
50 END
60 X=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]A[illegible]i [illegible]"),P1,P2):R
EM USE FOR ANY LENGTH.
```

PROGRAM.006 Text: Lower Case To Upper Case

The following program does just the opposite of PROGRAM.005, it changes lower case letters to UPPER case. It uses the same two parameters as PROGRAM.005

P1 Starting address of memory to change.
P2 Length.

```
0 REM PROGRAM.006
10 DIM A$(24)
20 A$="lower case to upper case"
30 X=USR(ADR("hh[illegible]"),ADR(A$),LEN(A$))
40 ? A$
50 END
60 X=USR(ADR("hh[illegible]"),P1,P2):RE
M USE FOR ANY LENGTH.
```

PROGRAM.007 Text: Normal To Inverse

This program changes normal text characters to inverse text characters in the memory range specified. The address and length of the memory to change are the required parameters.

P1 Starting address of memory to change.
P2 Length.

```
0 REM PROGRAM.007
10 DIM A$(22)
20 A$="NORMAL TO INVERSE TEXT"
30 X=USR(ADR("hh[illegible]"
),ADR(A$),LEN(A$))
40 ? A$
50 END
60 X=USR(ADR("hh[illegible]"),P1,P2):REM
USE FOR ANY LENGTH.
```

PROGRAM.008 Text: Inverse To Normal

The next program does just the opposite of PROGRAM.007. It changes inverse text to normal text in the memory range specified. The address and length of the memory to change are again the required parameters.

P1 Starting address of memory to change.
P2 Length.

```
0 REM PROGRAM.008
10 DIM A$(22)
20 A$="[illegible]"
30 X=USR(ADR("hh[illegible]h[illegible]hh[illegible]) [illegible]"
),ADR(A$),LEN(A$))
40 ? A$
50 END
60 X=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]) [illegible]
[illegible]) [illegible]"),P1,P2):REM
USE FOR ANY LENGTH.
```

PROGRAM.009 Text: Normal To Inverse; Inverse To Normal

The following program is a combination of PROGRAM.007 and PROGRAM.008. It changes normal text to inverse text and vice versa at the same time. Again, it uses the same two parameters as the last two programs.

P1 Starting address of memory to change.
P2 Length.

```
0 REM PROGRAM.009
10 DIM A$(49)
20 A$="NORMAL TEXT [illegible] CHANGED TO[illegible]
[illegible]AND[illegible]!"
30 X=USR(ADR("hh[illegible]h[illegible]hh[illegible]
[illegible]) [illegible]"),ADR(A$),LEN(A$))
40 ? A$
50 END
60 X=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]
[illegible]) [illegible]) [illegible]
[illegible]"),P1,P2):REM USE FOR ANY LENGTH.
```

Chapter 4
Fun With Memory

PROGRAM.010 Fill Memory

This program fills a memory area with your choice of a number from 0 to 255. If a string is filled and then printed to the screen the ATASCII character of that number will be shown. If you fill a text screen (GRAPHICS 0,1,or 2) memory area with a number the Display Character will be shown (See Appendix). Before filling any strings make sure they are set in memory, line 20, if encountered for the first time (See Introduction).

The sample program shows the filling of a string, A$, with the number 42 which is the ATASCII value for the asterik character. Three parameters are required.

P1 Starting address of memory to fill.
P2 Length.
P3 Number to place into memory.

```
0 REM PROGRAM.010
10 DIM A$(100)
20 A$=" ":A$(100)=A$:A$(2)=A$
30 X=USR(ADR("hh[illegible]h[illegible]hh[illegible]hh[illegible]"),
ADR(A$),LEN(A$),42):REM MAXIMUM LENGTH
   IS 256.
40 ? A$
50 END
60 X=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]hh[illegible]
[illegible]"),P1,P2,P3):REM *
   USE FOR ANY LENGTH OF MEMORY.
```

PROGRAM.011 Compare Two Memories

This program compares two areas of memory and puts a zero in the Return Variable if the two memory areas are equivalent. If the two memory areas are not equivalent then the Return Variable will contain the location within the memory areas where the first difference occurs.

The sample program shows two strings which are made equal except for the 77th character (lines 20 & 30). The machine routine works for any length of memory. Three parameters are required.

P1 Starting address of one memory area.
P2 Starting address of other memory area.
P3 Length of memory to search through.

```
0 REM PROGRAM.011
10 DIM A$(100),B$(100)
20 A$="X":A$(100)=A$:A$(2)=A$
30 B$=A$:B$(77)="Y"
40 LOC=USR(ADR("hh[illegible]
[illegible]
[illegible]"),ADR(A$),ADR(B$),LEN(A$))
50 ? LOC
```

PROGRAM.012 Search Memory

Have you ever wanted to search through a string or memory area for a certain word or number? This next program does just that. The Return Variable will contain the first location within the memory searched where the desired information is found. If the Return Variable contains a zero then the desired information was not found. This routine is good for any length of memory. You could search disk data by first bringing up a file (PROGRAM.036) or a Sector of data (PROGRAM.037) before using this routine. It is important to place the exact information, in character code, you are searching for in the first parameter string, otherwise, it may not be found or the incorrect information may be found.

The sample program below will return a value of 25 in the Return Variable 'LOC' (standing for Location) in line 60. The word 'GOOD' starts at the 25th memory position in the string A$. Four parameters are required.

P1 Starting address of the string holding the characters to be found.
P2 Number of characters to be found.
P3 Starting address of the main memory to search through.
P4 Length of the main memory in P3.

```
0 REM PROGRAM.012
10 DIM A$(72),FIND$(4),P$(93)
20 A$="NOW IS THE TIME FOR ALL GOOD ME
N TO COME TO THE AID OF THEIR COUNTRY"
30 FIND$="GOOD"
40 P$="hh[illegible]h[illegible]hh[illegible]h[illegible]h[illegible]h[illegible]h8[illegible]
[illegible]
[illegible]"
50 LOC=USR(ADR(P$),ADR(FIND$),LEN(FIND
$),ADR(A$),LEN(A$))
60 ? LOC
```

PROGRAM.013 Clearing Text Screens

The three short machine routines below will clear and blank the screen in the three text modes, GRAPHICS 0, 1, and 2. No parameters are required. Line 10 will clear a GRAPHICS 0 screen. To clear a GRAPHICS 1 screen, use line 30, for a GRAPHICS 2 screen, use line 40.

```
0 REM PROGRAM.013
10 X=USR(ADR("h[illegible]Y[illegible]X[illegible]
[illegible]")):REM CLEAR GRAPHICS 0.
20 GOTO 20
30 X=USR(ADR("h[illegible]Y[illegible]X[illegible]
[illegible]")):REM USE TO CLEAR A
   GRAPHICS 1 SCREEN.
40 X=USR(ADR("h[illegible]Y[illegible]X[illegible]"))
:REM USE TO CLEAR A GRAPHICS 2 SCREEN.
```

PROGRAM.014 Fill Graphics 0 Screen

The routine in line 10 fills a GRAPHICS 0 screen with a single character. The only parameter required is the Display

Number for the character to fill the screen (See Appendix). Line 20 just prevents the return of our program to the screen editor, so the cursor and 'READY' prompt won't appear.

P1 Display number of character to fill screen.

```
0 REM PROGRAM.014
10 X=USR(ADR("h[illegible]Y[illegible]X[illegible]hh[illegible]
[illegible]"),3):REM THE NUMBER '3' IS
   THE CODE FOR '#' IN DISPLAY MEMORY.
20 GOTO 20
```

PROGRAM.015 <u>Fill Graphics 1 Or 2 Screen</u>

To fill a GRAPHICS 1 text screen with a character, use the machine routine in line 20. For a GRAPHICS 2 text screen, use the routine in line 40. Unlike the GRAPHICS 0 screen, all the text characters are not readily available. Register 756 controls which half of the characters are available for use. The default value of memory 756 contains 224 and allows the use of the upper case, numbers, and punctuation characters. POKEing 756 with 226 allows lower case and graphic characters.

P1 Display number of character to fill screen.

```
0 REM PROGRAM.015
10 GRAPHICS 1+16
20 X=USR(ADR("h[illegible]Y[illegible]X[illegible]hh[illegible]
[illegible]"),3)
30 GOTO 30
40 X=USR(ADR("h[illegible]Y[illegible]X[illegible]hh[illegible]"),
P1):REM FOR A GRAPHICS 2 SCREEN.
```

PROGRAM.016 <u>Alternating Screen Fill: Graphics 0</u>

If you want to fill a GRAPHICS 0 screen with two alternating characters, use this machine language routine. Again, make sure you use the Display value of the characters you want. The two characters to place on the screen are the two required parameters.

P1 Display number of the first character on the screen.
P2 Display number of the second character on the screen.

```
0 REM PROGRAM.016
10 X=USR(ADR("hY[illegible]X[illegible]hh[illegible]hh[illegible]
[illegible]
[illegible]"),10,11)
20 GOTO 20
```

PROGRAM.017 Alternating Screen Fill: Graphics 1

Use this routine to put two alternating characters on a GRAPHICS 1 screen. Refer to PROGRAM.015 for further instructions on selecting the characters. The parameters are the same as for PROGRAM.016.

```
0 REM PROGRAM.017
10 GRAPHICS 1+16
20 X=USR(ADR("hY[illegible]X[illegible]hh[illegible]hh[illegible]
[illegible]
[illegible]"),10,11)
30 GOTO 30
```

PROGRAM.018 Alternating Screen Fill: Graphics 2

To put two alternating characters on a GRAPHICS 2 screen, use this routine. Refer to PROGRAM.015 for further instructions on selecting the proper set of characters to show on the screen. The parameters are the same as for PROGRAM.016.

```
0 REM PROGRAM.018
10 GRAPHICS 2+16
20 X=USR(ADR("hY[illegible]X[illegible]hh[illegible]hh[illegible]
[illegible]"),10,11)
30 GOTO 30
```

PROGRAM.019 Line Fill

The next program really contains three machine routines in one. It will fill a GRAPHICS 0, 1, or 2 screen with a single character between two specified lines. The variable 'M' in the program below is a multiplying factor, and also a parameter that adjusts the routine for the graphics mode being used. For a GRAPHICS 0 screen M=40 and the lines on the screen range from 0 to 23. For a GRAPHICS 1 screen M=20 and the lines also range from 0 to 23. For the GRAPHICS 2 screen M=20 and lines range from 0 to 11.

Lines 10 to 50 asks you for the GRAPHICS mode and sets up the correct value of 'M'. Four parameters are required.

- P1 Starting line number (*M)
- P2 Ending line number (*M)
- P3 The multipyling factor, M.
- P4 The character to Display.

```
0 REM PROGRAM.019
10 ? "[illegible]":? :? "WHAT GRAPHICS MODE (0,1
,2)";:INPUT CHOICE
20 IF CHOICE=0 THEN GRAPHICS 0:M=40:PO
KE 752,1:? :GOTO 60
30 IF CHOICE=1 THEN GRAPHICS 1+16:M=20
:GOTO 60
40 IF CHOICE=2 THEN GRAPHICS 2+16:M=20
:GOTO 60
50 GOTO 10
60 X=USR(ADR("[illegible]hh[illegible]eY[illegible]h[illegible]eX[illegible] [illegible]h8
[illegible]h[illegible]hh[illegible]e[illegible] [illegible]8[illegible] [illegible]hh[illegible] [illegible]
[illegible]"),5*M,9*M,M,4)
70 GOTO 70
80 REM LINE 60 FILLS LINES 5 TO 9
   INCLUSIVE.  THE DISPLAY VALUE OF
   '4' IS '$'.
```

Chapter 5
Numeric Arrays

Arrays are a systematic way of naming a large number of variables. Instead of giving pieces of data different variable names, all the pieces are grouped together and given one variable name. The individual pieces of data are identified by a position indicator, called an index. Remember to start with 0 as your first index counter, and use only positive integer values in the array, or the routines in this book won't work correctly.

Before using a numeric array you must specify the maximum length of the array in a DIM statement, just like in DIMensioning an ATARI string. ATARI BASIC can have one or two indexes which represent a one or two dimensional array. The programs and machine routines in this book are all for a one dimension array.

To find the starting location of a string in ATARI BASIC the ADR(string$) function is used. However, there is no identical statement for finding the starting address of a numeric array and a series of complicated steps must be performed. Once this address is found, the data can be manipulated. There are further complications to using the data in numeric arrays, which we'll discuss in PROGRAM.022.

PROGRAM.020 Starting Address

The next BASIC program and machine routine finds the starting address of a numeric array, and you'll need it to run the other programs in this section. Consequently, to run the other programs first 'LOAD' this program and then 'ENTER' the other programs into this one.

The numeric array variable name in this example is called 'ABC'. It is first DIMensioned in line 10 along with the string variable name 'NAME$' which, is set equal to the array name 'ABC' in line 30. Line 20 fills our array with 20 random numbers. Line 30 then places the ATASCII code value of each character of the array variable name 'ABC' into PAGE 6 (must start at 1536). This is done so the machine language routine can use direct addressing of the array name to check for its location in the variable table.

The actual machine language routine, P$, is called in line 60. The Return Variable, ADDRESS, will contain the starting address of the numeric array 'ABC'. The single parameter in the USR call is the length of the name of the array.

P1 The length of the name used for the numeric array.

```
0 REM PROGRAM.020
10 DIM P$(183),ABC(20),NAME$(3)
20 FOR I=0 TO 19:ABC(I)=INT(100*RND(0)
):NEXT I
30 NAME$="ABC":FOR I=1 TO LEN(NAME$):P
OKE 1535+I,ASC(NAME$(I,I)):NEXT I
40 P$(1)="hhh[illegible]
[illegible]
[illegible]"
50 P$(102)="[illegible]
[illegible]
[illegible]"
60 ADDRESS=USR(ADR(P$),LEN(NAME$))
70 ? ADDRESS
```

Now that we can find the starting address of an array (The zero index value), it would appear that we could just PEEK the array locations to find our numbers. But no matter how hard you try, you'll find the numbers just don't make sense. The problem lies in the fact that array numbers are stored in a six byte format called Binary Coded Decimal. This format allows precise mathematical calculations, but makes direct manipulation of the numbers difficult.

PROGRAM.021 will convert an integer array number into its decimal equivalent.

PROGRAM.021 Array Number To Decimal Equivalent

This program converts an integer Binary Coded Decimal (BCD) number in a numeric array into its decimal equivalent. ENTER this program into PROGRAM.020 and you will find the Return Variable 'NUMBER' contains the first number (zero index) in the array generated by PROGRAM.020. To get the next number in the array, you must move up in memory 6 bytes

(Remember, each BCD number takes 6 bytes of memory). This routine uses two internal operating system ROM routines to perform the conversion. The single parameter required by this routine is the starting address of the integer BCD number you wish to convert to decimal form. Of course, it is easier to use BASIC to retrieve the array values, but we wanted to show that the address arrived at in PROGRAM.020 was correct. Check the result of PROGRAM.021 by looking at the first array number using the BASIC statement '? ABC(0)'.

P1 Starting address of an integer array number.

```
0 REM PROGRAM.021
70 NUMBER=USR(ADR("hh[illegible]h[illegible] [illegible] [illegible]◆"),AD
DRESS):? NUMBER
```

PROGRAM.022 High Value Search

This program finds the highest value number in an integer array. Line 70 is the machine routine that searches the integer array and puts the highest value in the Return Variable. ENTER this program into PROGRAM.020. Two parameters are required.

P1 Address of the first array number to begin the search.
P2 The number of array numbers to search through.

Line 80 prints the entire array of the 20 random numbers from BASIC for proof that the machine routine has found the highest value.

```
0 REM PROGRAM.022
70 HIGH=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible] [illegible] [illegible]
[illegible]0 #[illegible]i [illegible] [illegible] [illegible] [illegible]
[illegible]◆"),ADDRESS,20)
80 FOR I=0 TO 19:? ABC(I):NEXT I:? "HI
GH= ";HIGH
```

PROGRAM.023 Low Value Search

This program is the same as PROGRAM.022, except it finds the lowest number in an integer array. It uses the same parameters as PROGRAM.022.

```
0 REM PROGRAM.023
70 LOW=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]
[illegible]0%[illegible]i[illegible]
[illegible]"),ADDRESS,20)
80 FOR I=0 TO 19:? ABC(I):NEXT I:? "LO
W= ";LOW
```

PROGRAM.024 Sort in Ascending Order

If you want to sort an integer array in ascending order, just ENTER the following routine into PROGRAM.020. The machine routine, D$, is too long for one line, so it must be placed in a string. Line 100 prints out the newly organized numeric array. The same two parameters from the last two programs are used.

- P1 Address of the first array number to begin the search.
- P2 The number of array numbers to search through.

```
0 REM PROGRAM.024
10 DIM P$(183),D$(151),ABC(20),NAME$(3
)
70 D$(1)="hh[illegible]h[illegible]h[illegible]h[illegible]
[illegible]0P[illegible]i[illegible]
[illegible]8[illegible]h"
80 D$(102)="[illegible]
[illegible]"
90 X=USR(ADR(D$),ADDRESS,20)
100 FOR I=0 TO 19:? ABC(I):NEXT I
```

PROGRAM.025 Sort In Descending Order

This program is the same as PROGRAM.024 except that it sorts the array in descending order.

```
0 REM PROGRAM.025
10 DIM P$(183),D$(159),ABC(20),NAME$(3
)
70 D$(1)="hh[illegible]"
80 D$(102)="[illegible]"
90 X=USR(ADR(D$),ADDRESS,20)
100 FOR I=0 TO 19:? ABC(I):NEXT I
```

PROGRAM.026 Search For A Value

To search through an integer array for a specfic number, use this next program. ENTER it into PROGRAM.020. In the sample program below we'll search for the number 33 in the 20 number array 'ABC'. Line 70 just makes sure the number 33 is in the random array at an index value of 12. The Return Variable in line 100 gives the Index value of the first occurance of the desired number in the array. If the number does not appear in the array the Return Variable returns the number 65535. Four parameters are required. Note that the first parameter is not the same as the first parameter in the other array routines. It must always be the starting address of the array, Index 0.

P1 Starting address of the array.
P2 The integer number to be located in the array.
P3 The Index number to begin the search.
P4 The last Index number to be searched.

```
0 REM PROGRAM.026
10 DIM P$(183),D$(120),ABC(20),NAME$(3
)
70 ABC(12)=33
80 D$(1)="hh[illegible]"
90 D$(102)="[illegible]"
100 INDEX=USR(ADR(D$),ADDRESS,33,0,19)
110 ? "DESIRED NUMBER AT INDEX ";INDEX
```

PROGRAM.027 Sum Array

The machine routine in line 70 will sum the contents of an integer array.The sum can not be over 65535. Again, ENTER this program into PROGRAM.020. Two parameters are required.

P1 Address of the first array number to begin the search.
P2 Starting with the array number at the address P1, the number of array numbers to search through.

```
0 REM PROGRAM.027
70 SUM=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]8[illegible]
[illegible]i[illegible] [illegible] [illegible] f[illegible]"),
ADDRESS,20)
80 ? SUM
```

PROGRAM.028 Sum Array, Unlimited

In order to find the sum of an integer array when the sum may be over 65535 use the next routine. The sum of the array is not in the Return Variable, but in the array element one Index value higher than the last value of the array. Therefore, make sure you DIMension the array at least one value higher than required. The same parameters are required as in PROGRAM.027.

```
0 REM PROGRAM.028
10 DIM P$(183),ABC(21),NAME$(3)
70 X=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]8[illegible]
[illegible]i[illegible] [illegible] [illegible] f[illegible]i[illegible]
[illegible]"),ADDRESS,20)
80 ? ABC(20)
```

Chapter 6
Graphics & Antic Modes

Atari computers have 17 different graphic modes; Antic modes 2-9 and A-F, and GTIA modes 9, 10, & 11. Refer to the Appendix for further information on Atari's Graphic modes. Graphics modes 0 through 11 are addressable from BASIC using the GRAPHICS command. A GRAPHICS mode call sets up a Display List in the computer (starting at PEEK(560)+PEEK(561)*256), which contains all the information necessary to set up the screen. If you have an XL or XE computer you can also call GRAPHICS modes 12, 13, 14, and 15 from BASIC, which represent Antic modes 4,5,C, and E (many times referred to 7.5). The remaining mode, Antic 3, is only obtainable by directly changing the Display List.

The following five programs give machine routines to quickly change the Display List to obtain Antic modes 3,4,5,C, and E. Now any Atari 400 or 800 owner can have all the modes at their disposal. Note that if you are going to write a BASIC program to pass around to your friends, to place on a BBS, or to market, and you're using Antic modes 4, 5, C, and E, it's a good idea use the routines in this book, instead of the XE or XL BASIC GRAPHICS call. Otherwise, the program will not work on an Atari 400 or 800.

PROGRAM.029 Antic Mode 3

Antic mode 3 is a text screen mode much like the Graphics 0 screen, but with 10 scan lines (horizontal lines) for each character. The Graphics 0 screen has 8 scan lines per character. Since there are two additional scan lines per character, the true descenders on the lower case letters j, p, q, and y can be designed along with true subscripts and superscripts. Since each character is 10 scan lines high, the screen will fit only 19 rows of characters instead of the normal 24 in Graphics mode 0. Of course, if you're into modifying Display Lists you could make more rows of characters beyond the normal screen border.

Designing a character set for this mode is tricky. The characters are still designed with 8 rows (one byte per row) but the computer will automatically add the two extra scan lines. For lower case letters, the extra two lines are added to the beginning of the character after the original two lines are moved to the end of the character. That's why some of the lower case letters in the program below are chopped off at the top, with the top appearing at the bottom. For all the graphic characters, numbers, and uppercase letters, the two extra lines are added to the bottom of the character.

A correct looking character set with true descenders is on the front of the program disk in the file "ANTIC3.SET". PROGRAM.038 in the Disk Input & Output section shows how to load and display this character set in conjunction with PROGRAM.029.

No parameters are required for the machine language routine in line 10, and it assumes the Display List is given by the normal locations in memory, 560 and 561.

```
0 REM PROGRAM.029
10 X=USR(ADR("h[illegible]/ [illegible]/ [illegible]1 [illegible]0 [illegible]
C[illegible]/[illegible] [illegible]0 [illegible]1 [illegible]/ [illegible]")
)
20 ? "JjPpQqYy"
```

PROGRAM.030 Antic Mode 4

Antic mode 4 (Graphics mode 12 on XL or XE) is a text mode screen with GRAPHIC mode 0 size text. The machine routine in line 20 changes the GRAPHICS 0 Display List into an Antic mode 4 Display List. This mode, along with Antic mode 5 can give multicolored individual text. The color and the shape of the character is produced from the data in the character set. Consequently, the normal Atari character set is almost unreadable in these modes.

Each byte of the character is taken as four bit pairs. Each of the four bit pair possibilites, 00, 01, 10, and 11, determine the color of two bits to be displayed on the screen. With this information you can really design some great looking screens in this graphics mode.

```
0 REM PROGRAM.030
10 X=USR(ADR("h[illegible]1 [illegible]0 [illegible]/ [illegible]/ [illegible]
D[illegible]/[illegible]/ [illegible]"))
20 LIST :REM SOMETHING TO LOOK AT.
```

PROGRAM.031 Antic Mode 5

The next program changes a GRAPHICS 1 screen into an ANTIC mode 5 screen (GRAPHICS 13 on an XL or XE computer). The same information for PROGRAM.030 applies here, except the characters are larger.

```
0 REM PROGRAM.031
10 GRAPHICS 1+16
20 ? #6;"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30 X=USR(ADR("h[illegible]1 [illegible]0 [illegible]/ [illegible]/ [illegible]
E[illegible]/[illegible]A[illegible]0 [illegible]1 [illegible]/ [illegible]")
)
40 GOTO 40
```

PROGRAM.032 Antic Mode C

The Antic mode C (GRAPHICS 14 on an XL or XE computer) screen is comparable to the Antic mode E screen, but displays only two colors. Each row requires 20 bytes of memory, and one half of the total memory of a ANTIC mode E screen. Line 30 draws a line across the screen for something to look at.

```
0 REM PROGRAM.032
10 GRAPHICS 7+16
20 X=USR(ADR("h[illegible]/ [illegible]/ [illegible]8[illegible]p[illegible]
[illegible]L[illegible]A[illegible]8[illegible]
[illegible]0 [illegible]1 [illegible]/ [illegible]"))
30 COLOR 1:PLOT 0,40:DRAWTO 79,40
40 GOTO 40
```

PROGRAM.033 Antic Mode E

Antic mode E (GRAPHICS 15 on an XL or XE computer) is one of the most popular and used color graphic mode screens. Micropainter, Atari and Koala Touch Tablets, and most other popular artist drawing programs use this mode. It is sometimes referred to as graphics mode 7.5 or 7+.

Each point on the screen is actually two pixels wide with four colors to choose from. Each color is determined by one of the four bit pairs possible in the display memory, 00, 01, 10, or 11. Antic mode E uses the same amount of memory as GRAPHICS mode 8. The following routine will be used by a number of the load and display picture screen programs in this book.

```
0 REM PROGRAM.033
10 GRAPHICS 8+16
20 COLOR 1:PLOT 0,0:DRAWTO 319,191
30 X=USR(ADR("h[illegible]"))
40 GOTO 40
```

PROGRAM.034 Graphics 8 Alternating Screen Fill

This program places two byte patterns on a GRAPHICS 8+16 screen in an alternating sequence. The two parameters required are two numbers from 0 to 255 whos binary image will be placed on the screen. Each number represents 8 sequencial dots on the screen in the pattern of the number's binary equivalent.

P1 First number.

P2 Second number.

```
0 REM PROGRAM.034
10 GRAPHICS 8+16:POKE 710,0:REM MAKE
   A BLACK BACKGROUND.
20 X=USR(ADR("h[illegible]"),8,255)
30 GOTO 30
```

PROGRAM.035 Clear Graphics 8, E, 9, 10, & 11 Screens

This program clears the above graphics screens by writing zero's to the screen memory. In the sample program, line 20 outlines a GRAPHICS 8 + 16 screen so we have something to erase. No parameters are required.

```
0 REM PROGRAM.035
10 GRAPHICS 8+16:POKE 710,0
20 COLOR 1:PLOT 0,0:DRAWTO 319,0:DRAWT
O 319,191:DRAWTO 0,191:DRAWTO 0,0:REM
DRAW SOMETHING ON SCREEN.
30 X=USR(ADR("h[illegible]Y[illegible]X[illegible]♥[illegible]←[illegible]
◆"))
40 GOTO 40
```

Chapter 7
Disk Input/Output

Section I

GENERAL LOAD & SAVE ROUTINES

Taking information from a disk or writing to the disk is another great use of machine language. Large amounts of data can be transferred quickly. This makes machine language very useful for picture loading and saving.

PROGRAM.036 Universal Loader/Saver

This program will load any amount of data from an Atari DOS disk file and place it anywhere in the computers memory. Conversely, it will also take any amount of data from the computers memory and save it to disk file. Using this machine program you can develop your own picture loading and saving formats and data transfer programs.

Four parameters are required. The first one, indicated by the variable 'CH', is the channel opened for communications to or from the disk. Line 20 opens channel #1. The second parameter tells the machine program whether it will read from or write to the disk. A '7' for this parameter means read from the disk. An '11' means write to the disk.

The variable 'RW', standing for Read/Write, is used in the next parameter, and it's also used to indicate if the opened channel in line 20 is for a read from (RW=4) or a write to the disk (RW=8).

The third parameter is the address in the computer where the data is to be read from or written to. The variable 'SD', standing for Source/Destination, is set to the first memory location of the screen memory as shown in line 40. The fourth parameter is the amount of consecutive data (bytes) to transfer. In the sample program below, 7680 bytes of data are to read from the disk file 'DATA.PIC' and placed into a

GRAPHICS 8 screen memory. Line 30 sets up the GRAPHICS 8+16 screen and changes the background color to black.

To take data from the computer and write it to the disk the only change required is to change the variable 'RW' to 8.

With this program you can transfer data to and from Atari strings, Page 6, or any location in memory.

P1 Channel opened.

P2 Read from disk=7, write to disk=11.

P3 Source or Destination of data in the computer memory.

P4 Amount of data to transfer.

```
0 REM PROGRAM.036
10 CH=1:RW=4:REM USE RW=8 FOR WRITE TO
   DISK.
20 CLOSE #1:OPEN #CH,RW,0,"D:DATA.PIC"
30 GRAPHICS 8+16:POKE 710,0
40 SD=PEEK(88)+PEEK(89)*256:REM THE
   LOCATION OF THE SCREEN MEMORY.
50 X=USR(ADR("hhh◣◣◣◣▣hh▣8┘h▣E┘h▣D┘h▣I
┘h▣H┘ V▣◆"),CH,RW+3,SD,7680):CLOSE #CH
60 GOTO 60
```

PROGRAM.037 Sector Loader/Saver

The last program reads data from or writes data to an Atari DOS file. This program reads data from or writes data to a sector on the disk. Atari sectors (One sector holds 128 bytes of data) in normal density are numbered from 1 to 720. In enhanced density the sectors are from 1 to 1024.

Four parameters are required for the machine routine in line 40. The first one is the source or destination of the 128 bytes of data to be transferred. In the sample program below the variable 'SD' in line 30 points to the address of the string 'S$' where the a sector of data from the disk is to be placed. This location could be any available consecutive area of computer memory. Page 6 is often used for this purpose.

The second parameter is the disk sector number. The third parameter is the disk drive number, 1-4, and the last parameter tells the machine routine whether data will be transferred to (RW=80) or from (RW=82) the disk.

Now you can develop your own BASIC program to read, edit, and save sector data.

In the sample below, we'll read the 10th sector on the disk in drive one and place the data in the string S$. Since the string S$ was not yet fixed in memory line 20 is required. POKEing memory address 766 with 1 in line 50 will prevent the screen editor from acting upon any of the normal display screen control functions (i.e., clear screen, cursor movement, etc.) that might appear in the S$ string (See Introduction for more information).

- P1 Source or Destination address for the data in the computer's memory.
- P2 Sector number on disk
- P3 Which disk drive, 1-4
- P4 Read from disk=82, write to disk=80.

```
0 REM PROGRAM.037
10 DIM S$(128)
20 S$=" ":S$(128)=S$:S$(2)=S$
30 RW=82:SD=ADR(S$):SECTOR=1:DRIVE=1:R
EM USE RW=80 FOR WRITE TO DISK.
40 X=USR(ADR("hh▄┐┘h▄┤┘h▄ ┴h▄◣┘hh▄┣hh
▄┣ S▣◆"),SD,SECTOR,DRIVE,RW)
50 POKE 766,1:? S$:POKE 766,0
```

PROGRAM.038 Character Set Loader

Although PROGRAM.036 could be used to perform the task of loading an alternate character set, the short routine below is tailored made for this duty. All you have to do is find an appropriate spot in memory to store the new character set (1024 bytes or 4 pages of memory) and run the machine program below.

Two parameters are required. The first one is the channel being opened to read from the disk, CH, and the second parameter is the address where the character set is to be stored.

In the sample below channel #1 is opened to read the character set from the file 'CHAR.SET' (located on the front of the program disk) in line 20. Line 10 looks at the high byte of the beginning of the screen's Display List, PEEK(561), and finds a location 4 pages back in memory, CB. This will be a safe place to store the new character set.

After the new character set has loaded, line 40 then switches to the new character set with POKE 756,CB. The default address of memory address 756 is 224 which is the standard Atari character set. You can place as many character sets in memory as room allows. The address of the new character set must be evenly divisible by 1024 for a GRAPHICS 0 screen and by 512 for a GRAPHICS 1 or 2 screen.

P1 Channel being opened, normally 1-5.
P2 Address to store new character set.

```
0 REM PROGRAM.038
10 CH=1:CB=PEEK(561)-4
20 CLOSE #CH:OPEN #CH,4,0,"D:CHAR.SET"
30 X=USR(ADR("hhh◣◣◣◣▩▩◥⬇B┘h⬇D┘h⬇E┘█H⬇
I┘█♥⬇H┘ V▤♦"),CH,CB):CLOSE #CH
40 POKE 756,CB
```

Section II

LOADING & SAVING PICTURE SCREENS

Loading and saving picture screen files from popular graphics creation programs on the market is a snap with the programs below. You can even make your own picture conversion programs by loading one picture from one format and then saving it in another format.

The general strategy of loading a picture is to first prepare the correct GRAPHIC screen, then call one of the loader programs. To save a picture screen you must first display the picture on the screen, then call the saving routine. It is very important to remember to always CLOSE the opened channel when all the data has been written, otherwise, you may loose your file.

PROGRAM.039 Magniprint Loader/Saver

Magniprint from ALPHA SYSTEMS is the most powerful screen printing program on the market. The program below will read or write a Magniprint format picture to the disk.

The first byte of data in Magniprint format is the graphics mode. A '14' indicates ANTIC mode E, a '9' indicates a GRAPHICS 9 picture, and a '24' indicates a GRAPHICS 8 picture. This first byte of data is best read (GET) from the disk or written (PUT) to the disk in BASIC. The machine program will read or write all the other data.

Magniprint format next contains the color data for registers 704-712, the actual Display List data, and then the picture data.

In the sample program below, we create a GRAPHICS 8 screen with a border, line 30, and then save it in Magniprint format. Line 10 opens a file on the disk, MAG.PIC, to receive the data and places the first data, 24, in the file to signify a GRAPHICS 8 screen.

If you were reading from the disk, 'RW' would equal 4, and you would have to GET the first data in BASIC and set up the correct GRAPHICS screen according to its value. If it was a '14' indicating a GRAPHICS E screen then PROGRAM.033 (GRAPHICS 15 for an XE or XL) would be required.

Two parameters are required. The first one is the channel being opened for the disk input/output. The second parameter tells the machine program whether data is to be read from the disk or written to the disk.

P1 Channel opened.
P2 Read from disk=7, Write to disk=11.

```
0 REM PROGRAM.039
10 RW=8:CH=1
20 OPEN #CH,RW,0,"D:MAG.PIC":PUT #CH,2
4
30 GRAPHICS 24:POKE 710,0:COLOR 1:PLOT
 0,0:DRAWTO 319,0:DRAWTO 319,191:DRAWT
O 0,191:DRAWTO 0,0
40 X=USR(ADR("hhh◣◣◣◣█hh█B┘██ ┘█H┘█ █E┘
███D┘ V██→█I┘█ ┘█H┘█1 █E┘█0 █D┘ V█◆"),C
H,RW+3):CLOSE #CH
```

PROGRAM.040 Micro-Painter Loader/Saver

The next program loads or saves a picture in Micro-Painter format. Micro-Painter format consists of the picture data first, 7680 bytes, then the color registers 712, 708, 709, and 710. Since Micro-Painter is an ANTIC E screen, PROGRAM.033 is used in line 30 to set up the correct graphics screen. To save a picture to disk, just make RW=8 and skip line 30.

The sample program will load a Micro-Painter picture from the file 'MICPNT.PIC' on the front of the program disk and display it on the screen. Three parameters are required. The third one is the source or destination (the variable 'SD') of the data in the computer memory, usually the display screen starting address PEEK(88)+PEEK(89)*256.

P1 Channel opened.
P2 Read from disk=7, Write to disk=11.
P3 Source or Destination of data in the computer memory.

```
0 REM PROGRAM.040
10 DIM P$(96):RW=4:CH=1:REM USE RW=8
   TO SAVE PICTURE TO DISK.
20 CLOSE #CH:OPEN #CH,RW,0,"D:MICPNT.P
IC"
30 GRAPHICS 24:A=USR(ADR("h[illegible]0 [illegible]1 [illegible]
[illegible]A[illegible]8[illegible]H[illegible]")):REM P
ROGRAM.033, GRAPHICS 24 TO ANTIC E.
40 SD=PEEK(88)+PEEK(89)*256
50 P$="hhh[illegible]hh[illegible]B[illegible]h[illegible]E[illegible]h[illegible]D[illegible]I[illegible]H[illegible]
 V[illegible]I[illegible]B[illegible] [illegible] | V[illegible] | V[illegible] | V[illegible] | V
[illegible] V[illegible] | V[illegible] | V[illegible] | V[illegible] [illegible]"
60 X=USR(ADR(P$),CH,RW+3,SD)
70 GOTO 70
```

PROGRAM.041 Graphic Master Loader/Saver

Loading and saving a Graphic Master picture is very similar to loading and saving a Micro-Painter picture. The only differences are that the Graphic Master picture is a GRAPHICS 8 picture, not ANTIC E, and the color data at the end of the file is for registers 708, 709, 710, and 712. Use the same parameters as for a Micro-Painter picture.

The program below is set up to load a Graphic Master picture. Just place the filename of one of your Graphic Master pictures in line 20 and run the program.

```
0 REM PROGRAM.041
10 CH=1:RW=4
20 CLOSE #1:OPEN #1,RW,0,"D:FILENAME"
30 GRAPHICS 24
40 SD=PEEK(88)+PEEK(89)*256
50 X=USR(ADR("hhh[illegible]hh[illegible]B[illegible]h[illegible]E[illegible]h[illegible]D[illegible]
I[illegible]H[illegible] V[illegible]H[illegible]I[illegible]E[illegible]D[illegible] V[illegible]"),C
H,RW+3,SD):CLOSE #CH
60 GOTO 60
```

PROGRAM.042 Fun With Art Loader

This program loads a Fun With Art picture from disk. Since an ANTIC E screen is required, line 30 is required. Two parameters are needed, the first being the channel opened for communication to the disk, and the second is the location to place the picture in the computer memory. The display screen starting address is normally chosen so the picture will be displayed upon loading.

P1 Channel opened.
P2 Starting address in computer memory to place picture data.

```
0 REM PROGRAM.042
10 DIM P$(143):CH=1
20 OPEN #1,4,0,"D:FILENAME"
30 GRAPHICS 24:A=USR(ADR("h[illegible]0 [illegible]1 [illegible]")):REM P
ROGRAM.033, GRAPHICS 24 TO ANTIC E.
40 SCN=PEEK(88)+PEEK(89)*256
50 P$(1)="hhh[illegible]B[illegible]I[illegible]H[illegible] V[illegible]
 V[illegible]K V[illegible] V[illegible] V[illegible] V[illegible] V[illegible]
[illegible]I[illegible]H[illegible]h[illegible]E[illegible]h[illegible]D[illegible] V[illegible]0 1[illegible]I"
60 P$(102)="[illegible]H[illegible] V[illegible]I[illegible]H[illegible]D[illegible]i
[illegible]D[illegible]E[illegible]i.[illegible]E[illegible] V[illegible]"
70 X=USR(ADR(P$),CH,SCN):CLOSE #CH
80 GOTO 80
```

PROGRAM.043 ComputerEyes Loader

ComputerEyes is a picture digitizing hardware and software system that can save pictures in either GRAPHICS 8, Antic E, or GRAPHICS 9 format. The saved picture file doesn't have a code to tell the computer what format it is, therefore, this information must be supplied by the user. The data format is the picture data only, 7680 bytes.

In the sample program below, the variable and first parameter 'TYPE' indicates the graphics screen to setup. For a High or Low Contrast capture, TYPE=0. You'll need an ANTIC E screen, and line 40 will load the picture. For a Normal, 4, or 8 Level capture, TYPE=1 and you'll use a GRAPHICS 8+16 screen with a black background (POKE 710,0), as shown in line 30. For a Graphics 9 capture, setup a GRAPHICS 9 screen in line 30 with TYPE=1.

Three parameters are required.

- P1 0 or 1 for the type of capture. See explaination above.
- P2 Channel opened.
- P3 Starting address in computer memory to place picture data.

```
0 REM PROGRAM.043
10 CH=1:TYPE=1:REM TYPE=0 FOR HIGH
   OR LOW CONTRAST CAPTURE(ANTIC E),
   OTHERWISE TYPE=1.
20 CLOSE #1:OPEN #1,4,0,"D:DATA.PIC"
30 IF TYPE=1 THEN GRAPHICS 24:POKE 710
,0:GOTO 50
40 GRAPHICS 24:A=USR(ADR("h[illegible]0 [illegible]1 [illegible]
[illegible]A[illegible]8[illegible]H[illegible]N[illegible]/[illegible]")):REM P
ROGRAM.033, GRAPHICS 24 TO ANTIC E.
50 SCN=PEEK(88)+PEEK(89)*256
60 X=USR(ADR("hhh[illegible]/[illegible]
[illegible]hh[illegible]N[illegible]B[illegible]h[illegible]E[illegible]h[illegible]D[illegible]I[illegible]H[illegible] V[illegible]"),
TYPE,CH,SCN):CLOSE #CH
70 GOTO 70
```

PROGRAM.044 Strip Poker Loader

Have you ever wanted to load a Strip Poker picture in your own BASIC program? Now you can with the routine below. Strip Poker files are specially coded to prevent 'cheap looks' but the machine routine below will decode them just after the picture loads. Again, an ANTIC E screen is required, as in line 30. Strip Poker pictures filenames are listed on the disk as 'OP1.X' or 'OP2.X' where X is a number representing the degree of undress.

P1 Channel opened.

P2 Screen address in computer memory to place picture data.

```
0 REM PROGRAM.044
10 DIM P$(114):CH=1
20 CLOSE #1:OPEN #1,4,0,"D:FILENAME"
30 GRAPHICS 24:A=USR(ADR("h[illegible]")):REM P
ROGRAM.033, GRAPHICS 24 TO ANTIC E.
40 SCN=PEEK(88)+PEEK(89)*256
50 P$(1)="hhh[illegible]"
60 P$(102)="[illegible]"
70 X=USR(ADR(P$),CH,SCN):CLOSE #1
80 GOTO 80
```

<u>PROGRAM.045</u> <u>Strip</u> <u>Poker</u> <u>Picture</u> <u>Saver</u>

With the next program you can now save pictures in Strip Poker format created by ComputerEyes (best in High or Low contrast capture) or by any of the other picture programs previously presented. Once the picture is brought up on the screen, jump to the program steps below to save it in Strip Poker format. Since Strip Poker uses only the top 73% of the screen (rows 0 to 139), you'll want to make sure the picture is positioned in that area of the screen. How are you going to do that? The answer lies in using the scrolling routines, PROGRAM.052.

The program below uses the same parameters as PROGRAM.044. Notice that we OPEN the channel number 1 in line 20 with an '8' instead of a '4' as in PROGRAM.044. A '4' means read from disk and an '8' means save to disk.

```
0 REM PROGRAM.045
10 DIM P$(120):CH=1
20 CLOSE #CH:OPEN #CH,8,0,"D:FILENAME"
30 SCN=PEEK(88)+PEEK(89)*256
40 P$(1)="hhh[illegible]B[illegible]h[illegible]E[illegible]h[illegible]D[illegible]
[illegible]I[illegible]H[illegible]E[illegible]
[illegible]E[illegible] V[illegible]I[illegible]H[illegible] | V[illegible] | V"
50 P$(102)="[illegible] | V[illegible] | V[illegible] V[illegible]"
60 X=USR(ADR(P$),CH,SCN):CLOSE #CH
```

PROGRAM.046 Typesetter Icon Loader

To load a Typesetter Icon to an ANTIC E screen, use the routine below. The Typesetter Icons that can be loaded by the routine below are the 15 sector file pictures created from the Sketch Pad portion of Typesetter.

You choose the colors for the picture. Line 40 not only finds the start of the screen memory, but also sets the background color to black. Note since the Typesetter Icon is only one forth the size of a full screen, you can experiment with the value of the variable 'SCN' below to move the Icon around the screen.

P1 Starting address in computer memory to place the picture.

P2 Channel opened.

```
0 REM PROGRAM.046
10 DIM P$(117):CH=1
20 OPEN #CH,4,0,"D:FILENAME"
30 GRAPHICS 24:A=USR(ADR("h[illegible]0 [illegible]1 [illegible]
[illegible]A[illegible]8[illegible]H[illegible]")):REM P
ROGRAM.033, GRAPHICS 24 TO ANTIC E.
40 SCN=PEEK(88)+PEEK(89)*256:POKE 710,
0
50 P$(1)="hh[illegible]h[illegible]hh[illegible]H[illegible]I
[illegible]D[illegible]E[illegible]B[illegible] V[illegible]
[illegible]i ([illegible]i [illegible]"
60 P$(102)="[illegible]d[illegible]"
70 BB=USR(ADR(P$),SCN,CH):CLOSE #CH
80 GOTO 80
```

Section III

KOALA PAD & ATARI TOUCH TABLET

These two drawing tablets are an excellent way to make and save pictures. They save a picture in a compressed format to save disk space. When a picture is to be saved, it's scanned twice in both a horizontal and a vertical direction, and then saved in the format requiring the least amount of memory. This compressed format is quite complicated and requires special routines to decipher. The next 4 programs, 47-50, present different ways to load a compressed picture and display it on the screen. Select the routine that is most appropriate to your needs.

Since Touch Tablet pictures are ANTIC E pictures, all the load routines contain PROGRAM.033. The last program, PROGRAM.051, will save a picture that is displayed on the screen into compressed format just the way the Touch Tablets do it.

PROGRAM.047 Load Horizontal Format Picture

The first 27 bytes of a compressed Touch Tablet picture contains information about the picture. The rest of the data is the picture. The 8th byte is the compression type, 1 for vertical compression and 2 for horizontal. If you plan on loading just one type of picture, you can use either this program or the next to save programming space over a dual loading routine. The only parameter required is which channel to open for reading the data from the disk(times 16).

The program below loads a horizontal compressed picture called 'HOR.PIC' from our program disk. A horizontal picture starts loading at the top of the screen and continues down the screen line by line. The machine code is contained in lines 40 through 60. Notice that the channel number in the parameter must be multiplied by 16.

P1 Channel opened multiplied by 16.

```
0 REM PROGRAM.047
10 DIM P$(249):CH=1
20 CLOSE #CH:OPEN #CH,4,0,"D:HOR.PIC"
30 GRAPHICS 24:A=USR(ADR("h[illegible]0 [illegible]1 [illegible]A[illegible]8[illegible]H[illegible]N[illegible]")):REM P
ROGRAM.033, GRAPHICS 24 TO ANTIC E.
40 P$(1)="hhh[illegible]B[illegible]I[illegible]E[illegible]H[illegible]D[illegible] V[illegible]h[illegible]H[illegible]E[illegible]D[illegible] V[illegible]D[illegible]E[illegible]H
[illegible] V[illegible]X[illegible]D[illegible]Y[illegible]E[illegible]I[illegible]H[illegible] V[illegible]0[illegible]"
50 P$(102)="[illegible]b[illegible]H[illegible] V[illegible]+ V
[illegible] V[illegible] V[illegible]
[illegible]e[illegible]) [illegible]H[illegible]E[illegible]D[illegible] V[illegible]H[illegible]"
60 P$(203)="[illegible] V[illegible] V[illegible]H[illegible]I[illegible]E[illegible]D
[illegible] V[illegible]H[illegible] e[illegible]I[illegible] e[illegible]"
70 X=USR(ADR(P$),CH*16):CLOSE #CH
80 GOTO 80
```

PROGRAM.048 Load Vertical Format Picture

This program is the mate to the last program. It loads a picture in vertical compression. A vertical picture starts loading from the left side of the screen and continues to the right side of the screen. Loading a vertical compressed picture is a little more complicated than a horizontal picture and takes a little more machine code. The sample program loads the picture 'VER.PIC' from the front of our program disk. It uses the same parameter as the last program.

```
0 REM PROGRAM.048
10 DIM P$(450):CH=1
20 CLOSE #CH:OPEN #CH,4,0,"D:VER.PIC"
30 GRAPHICS 24:A=USR(ADR("h[illegible]")):REM P
ROGRAM.033, GRAPHICS 24 TO ANTIC E.
40 P$(1)="hhh[illegible]"
50 P$(102)="[illegible]"
60 P$(203)="[illegible]"
70 P$(304)="[illegible]"
80 P$(405)="[illegible]"
90 X=USR(ADR(P$),CH*16):CLOSE #CH
100 GOTO 100
```

PROGRAM.049 Universal Loader

This program will load either a vertical or horizontal compressed picture. Consequently, the machine routine is much longer.

P1 Channel opened times 16.

```
0 REM PROGRAM.049
10 DIM P$(638):CH=1
20 CLOSE #1:OPEN #CH,4,0,"D:FILENAME"
30 GRAPHICS 24:A=USR(ADR("h[illegible]0 [illegible]1 [illegible]A[illegible]8[illegible]H[illegible]N[illegible]")):REM P
ROGRAM.033, GRAPHICS 24 ANTIC E.
40 P$(1)="hhh[illegible]"
50 P$(102)="[illegible]"
60 P$(203)="[illegible]"
70 P$(304)="[illegible]"
80 P$(405)="[illegible]"
90 P$(506)="[illegible]"
100 P$(607)="[illegible]"
110 X=USR(ADR(P$),CH*16):CLOSE #CH
120 GOTO 120
```

PROGRAM.050 Universal Loader, Short Form

This program will also load a Touch Tablet picture of either compression format, but the machine program was split up to require less string memory. Lines 40 to 60 contain a routine to check the type of compression. If it is horizontal, the loading process is completed by the USR call in line 70. The Return Variable 'X' in line 70 will contain the type of compression. If the picture was a vertical compression, the processing continues from line 70 to lines 80-120 where vertical decompression takes place. Notice that no parameter is required for the vertical decompression, line 120, since it is stored in an accesible memory location from the first machine program in line 70.

P1 Channel opened times 16. Only required for first machine program.

```
0 REM PROGRAM.050
10 DIM P$(393):CH=1
20 CLOSE #CH:OPEN #CH,4,0,"D:FILENAME"
30 GRAPHICS 24:A=USR(ADR("h[illegible]")):REM P
ROGRAM.033, GRAPHICS 24 TO ANTIC E.
40 P$(1)="hhh[illegible]"
50 P$(102)="H[illegible]"
60 P$(203)="[illegible]"
70 X=USR(ADR(P$),CH*16):IF X=2 THEN 13
0
80 P$(1)="h[illegible]"
90 P$(102)="[illegible]"
100 P$(203)="[illegible]
"
110 P$(304)="[illegible]"
120 X=USR(ADR(P$)):CLOSE #CH
130 GOTO 130
```

PROGRAM.051:Compressed Format Save

Now for the big program. This one will save any picture displayed on the screen in compressed Touch Tablet format. It saves the picture in the format which requires the least amount of memory. The routine is completely relocatable and does not require any of Page 6 to function. The only parameter required is the channel being opened to the disk for data transfer(times 16).

Lines 20-50 contains the machine routine that scans the picture vertically to determine the size of a vertical save. Then control passes to lines 70 through 90, which scan the picture for the size of a horizontal compression. The size of these two compressions are stored internally.

Line 100 contains the filename you want to call the compressed picture and OPENs up channel 1 for a write to the disk. Note that in order to read the picture using the Koala or Atari Touch Tablet software, you must use the '.PIC' extender.

The next machine routine in lines 110 through 130 writes the 27 byte header to the disk, and selects the format with the least memory by placing a 1 or 2 in the Return Variable 'TYPE' in line 130. If TYPE=1, a vertical compression is made as per lines 180 to 220. A horizontal compression, TYPE=2, uses the machine routine in lines 150 to 170. Line 230 completes the save to the disk.

Be very careful to renumber the lines correctly when using these routines in your own program.

The only parameter required is the channel opened for writing to the disk for the USR call in lines 130 and 230. Note the channel is not CLOSEd in line 130, since more writing to the disk is required in line 230.

P1 Channel opened times 16.

```
0 REM PROGRAM.051
10 DIM P$(486):CH=1
20 P$(1)="[illegible]"
30 P$(102)="[illegible]"
40 P$(203)="[illegible]"
50 P$(304)="[illegible]"
60 X=USR(ADR(P$)):REM SIZE VERTICAL
   COMPRESSION IN $CD & $CE.
70 P$(1)="[illegible]"
80 P$(105)="[illegible]"
90 X=USR(ADR(P$)):REM SIZE HORIZONTAL
   COMPRESSION IN $CB AND $CC.
100 CLOSE #CH:OPEN #CH,8,0,"D:NEW.PIC"
110 P$(1)="[illegible]"
120 P$(102)="[illegible]"
130 TYPE=USR(ADR(P$),CH*16):REM WRITE
    HEADER TO DISK
140 IF TYPE=1 THEN 180:REM VERTICAL
    COMPRESSION
145 REM HORIZONTAL COMPRESS
150 P$(1)="[illegible]"
160 P$(102)="[illegible]"
```

```
170 P$(203)="[illegible]":GOTO 230
180 P$(1)="[illegible]"
190 P$(102)="[illegible]"
200 P$(203)="[illegible]"
210 P$(304)="[illegible]"
220 P$(405)="[illegible]"
230 X=USR(ADR(P$),CH*16):CLOSE #CH
```

Chapter 8
Fun With Pictures

PROGRAM.052 Picture Scrolling

Once you have a GRAPHICS 8+16 or ANTIC E picture displayed on the screen, you can use the following routines to scroll it around the screen. The picture will wrap around the screen, not just run off the border. Now you can shift any picture to any position on the screen and resave it using the screen saving routines in this book.

The sample program below reads joystick zero (the first joystick) in lines 70 to 110 and passes the direction chosen to the proper machine scroll routine. Later, we'll show you machine routines to read the joystick. For fine scrolling to the left use line 180. For fine scrolling to the right use line 190. No parameters are required.

To demonstrate this routine, first load PROGRAM.040, the Micro-Painter screen loading routine, and ENTER this program into it.

```
0 REM PROGRAM.052
70 IF STICK(0)=14 THEN GOSUB 120:REM U
P
80 IF STICK(0)=13 THEN GOSUB 130:REM D
OWN
90 IF STICK(0)=7 THEN GOSUB 140:REM CO
ARSE RIGHT
100 IF STICK(0)=11 THEN GOSUB 150:REM
COARSE LEFT
110 GOTO 70
120 X=USR(ADR("h[illegible]"
)):RETURN
130 X=USR(ADR("h[illegible]")):RETURN
140 X=USR(ADR("h[illegible]")):RETURN
150 X=USR(ADR("h[illegible]")):RETURN
160 REM *
170 REM REPLACE LINE 140 WITH LINE
    180 AND LINE 150 WITH 190 FOR FINE
    SCROLLING LEFT AND RIGHT.
180 X=USR(ADR("h[illegible]")):RETURN
190 X=USR(ADR("h[illegible]")):RETURN
```

PROGRAM.053 Inverse Picture

The short routine below produces an inverse image of a picture on the screen, like a negative of a photograph. It replaces any zero's in the screen memory bytes with ones, and vice versa. Try this routine when copying a picture to a printer, since the results may turn out better.

Enter the program below into PROGRAM.040. Line 70 holds the picture on the screen for a short time before it is inverted. No parameters are required.

```
0 REM PROGRAM.053
70 FOR I=1 TO 500:NEXT I
80 X=USR(ADR("h[illegible]"))
90 GOTO 90
```

PROGRAM.054 Epson/Gemini Picture Printer

The routine below will print the picture on the screen (GRAPHICS 8+16 or ANTIC E), on an Epson or Gemini compatible printer. Lines 10 through 60 is PROGRAM.040, which brings up a Micro-Painter picture on the screen.

Once the picture is on the screen, line 70 opens a channel of communication, channel 2, to the printer. Line 80 sends the correct line spacing to the printer. The number 8 in line 80 requests 8/72 line spacing, some printers may require a different line spacing.

Lines 90 and 100 contain the machine code, which scans the picture and sends the appropriate dot image patterns to the printer. This routine needs 198 memory locations in a string or in Page 6 to function (third parameter). The routine below uses Page 6, starting at memory location 1536.

The routine will print the screen in either normal or inverse print (second parameter) and even checks for the two codes that can't be printed. One of these is the number 155, which is the End-Of-Line signal to the printer, and the other is two number 13's in a row. In the first case, 153 is substituted for 155, and in the second case, the second number 13 is changed to an 11. Four parameters are required for the USR call in line 110.

- P1 Channel opened to the printer.
- P2 0 for normal print, 1 for a inverse (negative image) print.
- P3 Address of a 198 byte memory for temporary use.
- P4 Starting picture memory address.

```
0 REM PROGRAM.054
10 DIM P$(163):RW=4:CH=1
20 CLOSE #CH:OPEN #CH,RW,0,"D:MICPNT.P
IC"
30 GRAPHICS 24:A=USR(ADR("h[illegible]0 [illegible]1 [illegible]
[illegible]A[illegible]8[illegible]H[illegible]N[illegible]")):REM P
ROGRAM.033, GRAPHICS 24 TO ANTIC E.
40 SD=PEEK(88)+PEEK(89)*256
50 P$="hhh[illegible]hh[illegible]B[illegible]h[illegible]E[illegible]h[illegible]D[illegible]I[illegible]H[illegible]
 V[illegible]I[illegible]B[illegible] | V[illegible] | V[illegible] | V[illegible] | V
[illegible] V[illegible] | V[illegible] | V[illegible] | V[illegible]"
60 X=USR(ADR(P$),CH,RW+3,SD)
70 CHP=2:CLOSE #CHP:OPEN #CHP,8,0,"P:"
80 ? #CHP;CHR$(27);CHR$(65);CHR$(8)
90 P$(1)="hhh[illegible]B[illegible]hh[illegible]h[illegible]E[illegible]h[illegible]D[illegible]
[illegible]K[illegible]h[illegible]h[illegible]i[illegible]i
[illegible]H[illegible]I[illegible]H[illegible]E[illegible]H[illegible]"
100 P$(102)="[illegible]H[illegible]
[illegible]8[illegible]([illegible] V[illegible]([illegible]"
:P$(95,95)=CHR$(155)
110 X=USR(ADR(P$),CHP,0,1536,SD):CLOSE
 #CHP
```

PROGRAM.055 Nec/C.Itoh Picture Printer

If you have an older NEC or C.Itoh compatible printer, you may need to use the following routine to print your picture (Some of the newer models have Epson compatibility, so try PROGRAM.054). Just ENTER this program into PROGRAM.054 for a demonstration. Line 80 is the proper line spacing statement for the NEC type printers, which is 16/144 inch spacing. The same parameters are used in this routine as for PROGRAM.054.

```
0 REM PROGRAM.055
80 ? #CHP;CHR$(27);CHR$(84);"16":REM L
INE SPACING
90 P$(1)="hhh[illegible]B[illegible]hh[illegible]h[illegible]E[illegible]h
[illegible]D[illegible]S[illegible]0[illegible]1[illegible]9[illegible]2[illegible]
h[illegible]h[illegible]H[illegible]I[illegible]E[illegible]H[illegible]"
100 P$(102)="[illegible]H[illegible]
[illegible]i([illegible] V[illegible]":
P$(98,98)=CHR$(155)
```

PROGRAM.056 Picture Fade In/Out

Here is a routine that will fade a picture into a blank screen or into another picture. At first glance, the program looks complicated because it requires three other programs in this book to function. In general, a picture is loaded into an area of memory that is not the display screen, then it is faded into the active display screen.

Line 10 does the necessary DIMension statements and, along with line 20, sets the display screen to ANTIC E, since we are going to load a Micro-Painter picture, PROGRAM.040. Line 30 points to an area of memory 31 pages (7936 bytes) less than the high byte of the active screen's Display List. This is a safe place to load the picture before it's faded into the active screen area. Lines 40 to 60 load the Micro-Painter picture (PROGRAM.040) into this specially set up area of memory.

In order to get some randomness to the fade routine, random number tables are setup. PROGRAM.093 is used to setup these tables in lines 70 to 100, 200, and 210. What happens is a string, S$, is setup, with the numbers ranging from 0 to 255, line 70-80, and these numbers are taken at random and placed into Page 6. Page 6 must be used for this table. Then the string, S$, is again used to set up 30 non-repeating random numbers (representing the 30 pages of a screen) into string T$.

Line 110 contains the machine code for the actual fade routine which is called in line 120. Eleven parameters are required.

P1 Starting address where hidden picture was placed.

P2 Address of the 30 random number string.

P3-P11 Special numbers used internally in the machine program. You can change these numbers for weird effects as long the last one is 0. Keeping 255 as the second to last one will assure the full picture is on the screen at the completion of the fade routine. All numbers must be from 0 - 255.

Once the picture loads there will be about an 8 second delay to set up the random tables before the picture begins to fade-in. Once the tables are generated, they don't have to be constructed again. Simply load another picture to the hidden memory and call the fade routine again in line 120. Now the picture just loaded will fade into the old picture.

```
0 REM PROGRAM.056
10 DIM P$(96),S$(256),FN$(17),T$(30):C
H=1:RW=4
20 GRAPHICS 24:A=USR(ADR("h[illegible]")):REM P
ROGRAM.033, GRAPHICS 24 TO ANTIC E.
30 SD=(PEEK(561)-31)*256
40 CLOSE #CH:OPEN #CH,RW,0,"D:MICPNT.P
IC"
50 P$="hhh[illegible]"
60 X=USR(ADR(P$),CH,RW+3,SD)
65 REM ** PICTURE DATA NOW LOADED INTO
   AN AREA OF MEMORY THAT IS NOT THE
   DISPLAY SCREEN AREA OF MEMORY.
70 S$=" ":S$(256)=S$:S$(2)=S$:SIZE=256
80 X=USR(ADR("hh[illegible]"),ADR(S$),1536,SIZE):GOSUB 200
90 T$=" ":T$(30)=T$:T$(2)=T$:SIZE=30
100 X=USR(ADR("hh[illegible]"),ADR(S$),ADR(T$),SIZE):GOSUB 200
105 REM ** RANDOM TABLES NOW MADE.
110 P$="hh[illegible]"
120 X=USR(ADR(P$),SD,ADR(T$),2,34,42,1
70,174,238,239,255,0)
130 GOTO 130
195 REM * SUBROUTINE FROM PROGRAM.093
200 FOR N=SIZE TO 1 STEP -1
210 X=USR(ADR("hhh[illegible]"),INT(N*RND(0))):NEXT N:RETUR
N
```

Chapter 9
Display list Interrupt Routines

Display List Interrupts (DLI) can only be obtained through the use of machine language. When an image is displayed on the TV or monitor screen, it's drawn line by line starting at the top of the screen and working its way to the bottom. Each line is drawn from left to right. When the scanning beam reaches the right end of a line it turns off, and must shift to the beginning of the next line. During this time the computer checks an address in memory, decimal locations 512 and 513, and jumps to the address indicated in these registers. The default address is simply an instruction that continues normal processing. However, if that address points to a users own machine code it will be executed.

In order to select the proper line on the screen to enable a DLI, the corresponding line in the Display List must have its bit 7 set. Once the proper setup is completed nothing will happen until the DLI is activated by placing 192 in memory location 54286.

All the routines in this book are set up in the string DLI$. You must use a string or Page 6 for these routines, since they can not be called directly by a USR call.

PROGRAM.057 Two Color Screen

One of the most popular uses of DLI's are to split the screen into two or more background colors. The following program will split a GRAPHICS 0 screen into two colors, the background color and one selected by you. The background color of a GRAPHICS 0 screen is color register 710.

Three parameters are required. The first one is the address of the DLI routine. This may seem redundant because we already use the address of the DLI routine in the USR call, but it is necessary since we need to find the location within the string, DLI$, that the actual DLI routine starts. The DLI routine actually contains two machine routines. Don't forget line 40, which activates the DLI routines.

P1 Starting address of the DLI machine code.
P2 Line number (1-23) to start the color change.
P3 Color desired (0-255) for second background color.

```
0 REM PROGRAM.057
10 DIM DLI$(61)
20 DLI$="h[illegible]1 [illegible]0 [illegible]h[illegible] h[illegible]i 2[illegible] [illegible] hh
[illegible]i[illegible]hh[illegible]H[illegible]h@"
30 X=USR(ADR(DLI$),ADR(DLI$),10,22)
40 POKE 54286,192
```

PROGRAM.058 Three Color Screen

This program is a 3 color split screen version of the last program. Five parameters are required.

P1 Starting address of the DLI machine code.
P2 First line number for first color change.
P3 Second line number for second color change.
P4 First color value (0-255).
P5 Second color value (0-255).

```
0 REM PROGRAM.058
10 DIM DLI$(86)
20 DLI$="h[illegible]1 [illegible]0 [illegible] h[illegible] h[illegible]i ?[illegible] [illegible]
hh[illegible]i[illegible]hh[illegible]hh[illegible]H[illegible]
[illegible]h@"
30 X=USR(ADR(DLI$),ADR(DLI$),10,15,8,2
2)
40 POKE 54286,192
```

PROGRAM.059 Four Color Screen

This program is a 4 color split screen routine. Seven parameters are required.

P1 Starting address of the DLI machine code.
P2 First line number for first color change.
P3 Second line number for second color change.
P4 Third line number for third color change.
P5 First color value (0-255).
P6 Second color value (0-255).
P7 Third color value (0-255).

```
0 REM PROGRAM.059
10 DIM DLI$(102)
20 DLI$="[illegible]"
30 X=USR(ADR(DLI$),ADR(DLI$),6,12,20,8
,22,46)
40 POKE 54286,192
```

PROGRAM.060 Multi-Color Screen

The following routine allows up to 23 color changes on a GRAPHICS 0 screen. The routine requires the line numbers and then the color values be placed into Page 6 starting at memory 1538 to function. Memory locations 1536 and 1537 are used internally by the machine program. Line 30 places 23 line numbers (1-23) into Page 6 starting at 1538. Line 40 does the USR call, which changes the Display List at the line numbers in the Page 6 memory. Now the color values (line 60) are placed into Page 6, starting at 1538, by line 50. The DLI routine will look into Page 6 for these color values. When it's time to activate the DLI routine and make all those color changes we simply POKE 54286 with 192 as per line 70.

Two parameters are required in the machine routine in line 40.

P1 Starting address of the DLI machine code.
P2 The number of line numbers to enact a color change.

```
0 REM PROGRAM.060
10 DIM DLI$(98)
20 DLI$="h[illegible]1 [illegible]0 [illegible]h[illegible] [illegible]h[illegible]i ?[illegible] [illegible] [illegible]hh
[illegible]H[illegible]
[illegible]h[illegible]h@"
30 FOR I=1 TO 23:POKE 1537+I,I:NEXT I:
LINES=23:REM FIRST PUT 23 LINE NUMBERS
   IN PAGE 6 STARTING AT 1538.
40 X=USR(ADR(DLI$),ADR(DLI$),LINES)
50 FOR I=0 TO 23:READ A:POKE 1538+I,A:
NEXT I:REM NOW PUT COLORS INTO PAGE 6
   STARTING AT 1538.
60 DATA 4,8,12,16,20,24,28,32,36,40,44
,48,52,56,60,64,68,72,76,80,84,88,92,9
6
70 POKE 54286,192
```

The following four programs are DLI routines for changing the background colors of GRAPHICS 1 or 2 screens. These screens are Atari's multicolor text mode screens. For GRAPHICS 1 you can change color on lines 1-23 and for a GRAPHICS 2 screen lines 1-11.

PROGRAM.061 Graphics 1 & 2, Two Color Screen

This program is the GRAPHICS 1 or 2 screen equivalent of PROGRAM.057.

P1 Starting address of the DLI machine code.
P2 Line number to start second color.
P3 Second color, 0-255.

```
0 REM PROGRAM.061
10 DIM DLI$(61)
20 GRAPHICS 1+16
30 DLI$="h[illegible]1 [illegible]0 [illegible]h[illegible] [illegible]h[illegible]i 2[illegible] [illegible] [illegible]hh
[illegible]i[illegible]hh[illegible]H[illegible]h@"
40 X=USR(ADR(DLI$),ADR(DLI$),10,22)
50 POKE 54286,192
60 GOTO 60
```

PROGRAM.062 Graphics 1 & 2, Three Color Screen

This program is the GRAPHICS 1 or 2 screen equivalent of PROGRAM.058. The same parameters apply.

```
0 REM PROGRAM.062
10 DIM DLI$(86)
20 GRAPHICS 1+16
30 DLI$="h[illegible]"
40 X=USR(ADR(DLI$),ADR(DLI$),10,15,8,2
2)
50 POKE 54286,192
60 GOTO 60
```

PROGRAM.063 Graphics 1 & 2, Four Color Screen

This program is the GRAPHICS 1 or 2 screen equivalent of PROGRAM.059. The sample program has a GRAPHICS 2 background screen. The same parameters apply.

```
0 REM PROGRAM.063
10 DIM DLI$(102)
20 GRAPHICS 2+16
30 DLI$="h[illegible]"
40 X=USR(ADR(DLI$),ADR(DLI$),3,6,9,8,2
2,46)
50 POKE 54286,192
60 GOTO 60
```

PROGRAM.064 Graphics 1 & 2, Multi-Color Screen

This program is the GRAPHICS 1 or 2 screen equivalent of PROGRAM.060. The same parameters and method of placing the line numbers and color values in Page 6 apply.

```
0 REM PROGRAM.064
10 DIM DLI$(98)
20 GRAPHICS 2+16
30 DLI$="h[illegible]1 [illegible]0 [illegible]h[illegible] [illegible]h[illegible]i ?[illegible] [illegible] [illegible]hh
[illegible]i[illegible]H[illegible]H[illegible]
[illegible]h[illegible]h@"
40 LINES=11:FOR I=1 TO LINES:POKE 1537
+I,I:NEXT I:REM FIRST PUT LINE NUMBERS
   IN TO PAGE 6 STARTING AT 1538.
50 X=USR(ADR(DLI$),ADR(DLI$),LINES)
60 FOR I=0 TO 23:READ A:POKE 1538+I,A:
NEXT I:REM NOW PUT COLORS INTO PAGE 6
   STARTING AT 1538.
70 DATA 4,8,12,16,20,24,28,32,36,40,44
,48,52,56,60,64,68,72,76,80,84,88,92,9
6
80 POKE 54286,192
90 GOTO 90
```

PROGRAM.065 Graphics 8 & E Multi-Color Screen

This program is the equivalent of PROGRAM.060 except it is for a GRAPHICS 8+16 or Antic E screen. For these screens color can be changed on lines 1-190. The sample program shows the placement of 10 random color changes (line 60) to a GRAPHICS 8 screen. Line 70 contains the line numbers where the color changes are to begin. The same parameters apply as for PROGRAM.060.

```
0 REM PROGRAM.065
10 DIM DLI$(107)
20 LINES=10:GRAPHICS 24
30 DLI$="h[illegible]
h@"
40 FOR I=1 TO LINES:READ A:POKE 1537+I
,A:NEXT I:REM LINE NUMBERS
50 X=USR(ADR(DLI$),ADR(DLI$),LINES)
60 FOR I=1 TO LINES:A=INT(256*RND(0)):
POKE 1537+I,A:NEXT I:REM COLORS
70 DATA 20,40,60,80,90,100,110,130,150
,170
80 POKE 54286,192
90 GOTO 90
```

PROGRAM.066 Two Character Sets

With the next short DLI routine you can place two different character sets on a GRAPHICS 0, 1, or 2 screen at the same time.

Lines 10 to 30 are from PROGRAM.038 which loads 'CHAR.SET', an alternate character set, from the front of our program disk to a safe area of memory. Line 10 loads the alternate character set to PEEK(561)-4 which is an address evenly divisible by 1024 (See PROGRAM.038 for details). Line 50 then sets up the DLI routine in line 40 and line 60 turns it on. Remember, you can change lines 1-23 on a GRAPHICS 0 & 1 screen and lines 1-11 on a GRAPHICS 2 screen. Three parameters are required.

- P1 Starting address of the DLI machine code.
- P2 Line number to begin the alternate character set.
- P3 Starting address of the alternate character set in the computer memory.

```
0 REM PROGRAM.066
10 DIM DLI$(61):CB=PEEK(561)-4
20 CH=1:CLOSE #CH:OPEN #CH,4,0,"D:CHAR
.SET"
30 X=USR(ADR("hhh[illegible]B[illegible]h[illegible]D[illegible]h[illegible]E[illegible]H[illegible]
I[illegible]H[illegible] V[illegible]"),CH,CB):CLOSE #CH:REM ***
   PROGRAM.038, LOADS CHARACTER SET.
40 DLI$="h[illegible]1 [illegible]0 [illegible]h[illegible] h[illegible]i 2[illegible] [illegible] hh
[illegible]i[illegible]hh[illegible]H[illegible]h@"
50 X=USR(ADR(DLI$),ADR(DLI$),10,CB)
60 POKE 54286,192
```

<u>PROGRAM.067 Insert Alternate Character Set</u>

This program will change 1 or more lines to an alternate character set on a GRAPHICS 0,1, or 2 screen. This way you can insert the alternate character set in only a few lines. The program is like the last one except a third parameter is required. The sample program inserts six lines of the alternate character set in the middle of the screen. Line 5 was added to write something to the GRAPHICS 1 screen. Note, that in line 10 the alternate character set was loaded to PEEK(561)-5 to obtain an address evenly divisible by 512 (48K memory). Refer to PROGRAM.038 for more details.

P1 Starting address of the DLI machine language routine.
P2 Line number to begin the alternate character set.
P3 Line number to return to the the normal character set.
P4 Starting address of the alternate character set.

```
0 REM PROGRAM.067
5 GRAPHICS 1+16:FOR I=0 TO 23:? #6;"
Machine Language":NEXT I
10 DIM DLI$(95):CB=PEEK(561)-5
20 CH=1:CLOSE #CH:OPEN #CH,4,0,"D:CHAR
.SET"
30 X=USR(ADR("hhh[illegible]B[illegible]h[illegible]D[illegible]h[illegible]E[illegible]H[illegible]
I[illegible]H[illegible] V[illegible]"),CH,CB):CLOSE #CH:REM ***
   PROGRAM.038, LOADS CHARACTER SET.
40 DLI$="h[illegible]1 [illegible]0 [illegible]h[illegible] h[illegible]iF[illegible] [illegible] hh
[illegible]i[illegible]hh[illegible]i[illegible]hh[illegible]
[illegible]H[illegible]h@[illegible]h@"
50 X=USR(ADR(DLI$),ADR(DLI$),8,14,CB)
60 POKE 54286,192
70 GOTO 70
```

PROGRAM.068 Flip Text

If you want to flip the text beginning at some line on the screen then use the next routine. This flipping of characters is useful when displaying the bottom half of cards using redefined characters. The routine below works for a GRAPHICS 0, 1, or 2 screen.

The sample program shows the flipping of text on a GRAPHICS 0 screen starting at line 10.

P1 Starting address of the DLI machine code.
P2 Line number to begin text flip.

```
0 REM PROGRAM.068
10 DIM DLI$(57)
20 DLI$="h[illegible]1 [illegible]0 [illegible]h[illegible] [illegible]h[illegible]i .[illegible] [illegible] [illegible]hh
[illegible]H[illegible] [illegible]i[illegible] [illegible]H[illegible]H[illegible]H[illegible]h@"
30 X=USR(ADR(DLI$),ADR(DLI$),10)
40 POKE 54286,192
```

PROGRAM.069 Insert Flipped Text

This program will demonstrate flipping text within a group of lines chosen by the user. The routine works for a GRAPHICS 0, 1, or 2 screen.

P1 Starting address of the DLI machine code.
P2 Line number to begin text flip.
P3 Line number to return to upright text.

```
0 REM PROGRAM.069
10 DIM DLI$(93)
20 DLI$="h[illegible]1 [illegible]0 [illegible]h[illegible] [illegible]h[illegible]i D[illegible]♥ [illegible] [illegible]hh
[illegible]H[illegible] [illegible]i[illegible] [illegible]hh[illegible]H[illegible] [illegible]i[illegible] [illegible]
♦H[illegible] [illegible] [illegible]h@[illegible]h@"
30 X=USR(ADR(DLI$),ADR(DLI$),10,13)
40 POKE 54286,192
```

PROGRAM.070 Clear DLI Routines

Now that we have all those fancy DLI routines, we need a way of removing them without using the almighty RESET key! The listing below shows the correct routine to use for a GRAPHICS 0, 1, 2, 8, or ANTIC E screen. Just insert the appropriate routine into your BASIC program where you want the DLI to end. No parameters are required.

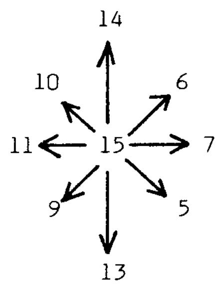
```
0 REM PROGRAM.070
10 REM ** CLEARS DLI FOR GRAPHICS 0
   OR 1 SCREEN.
20 X=USR(ADR("h[illegible]1 [illegible]0 [illegible]) ▶[illegible])
▶[illegible]♥ [illegible] [illegible]♦"))
30 REM
40 REM ** CLEARS DLI FOR GRAPHICS 2
   SCREEN.
50 X=USR(ADR("h[illegible]1 [illegible]0 [illegible]) ▶[illegible])
▶[illegible]♥ [illegible] [illegible]♦"))
60 REM
70 REM ** CLEARS DLI FOR GRAPHICS 8 OR
   ANTIC E SCREEN.
80 X=USR(ADR("h[illegible]1 [illegible]0 [illegible]) ▶[illegible])
▶[illegible]d[illegible]f[illegible]) ▶[illegible]♥ [illegible] [illegible]♦"
))
```

Chapter 10
Joystick Routines

Reading the joystick is not difficult in BASIC. To read the value of the first joystick, Joystick 0, try the following BASIC line.

```
10 ? STICK(0):GOTO 10
```

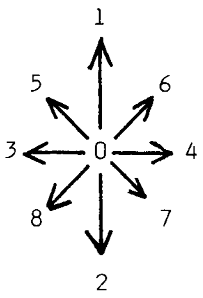
Depending on the position of the joystick, the following numbers will appear.



Normal
Joystick
Response

Normally, when the joystick position is read some action will be taken. This process takes a mess of IF...THEN statements in BASIC. If the joystick numbers would have been assigned as 0-8 then the powerful BASIC single statement "ON...GOTO" or "ON...GOSUB" could be used for all the decision making. The joystick values may not make too much sense in BASIC but they are perfect for machine language reading of the joystick position.

The joystick machine routines in this section will change the joystick positions to the following values so the "ON...GOTO" or "ON...GOSUB" statement can be used.



Redefined
Joystick
Response

Here are some important points to remember. The first joystick port is Joystick 0, the second joystick port is Joystick 1, etc.. Only the Atari 400 and 800 are capable of addressing Joysticks 2 and 3 so don't use any of those routines on your XE or XL.

Make sure you place the proper number of line numbers following the ON...GOTO or ON...GOSUB statement to prevent an error from occurring. For example, if you only want to use the up and down position of PROGRAM.071, you still must include the two line numbers for the left and right position in the ON...GOTO or ON...GOSUB statement. Just use the line number that sends the program back to the joystick read routine.

Lastly, the neutral position of the joystick will return a value of zero. The ON...GOTO or ON...GOSUB statements are programmed to ignore this number and processing will continue to the next BASIC line.

PROGRAM.071 Joystick 0 or 1 - 5 Position Read

Since the majority of programs written use Joystick 0 and only the neutral, up, down, left, and right positions, the following short routine was developed. Line 10 is the machine code and USR call statement. The Return Variable, 'X' in our example, contains the position of the joystick in our new numbering system (0-4). Line 20 is the ON...GOTO statement which branches control of the program according to the position of the joystick.

The sample program prints out the position of Joystick 0. For Joystick 1 replace the code in line 10 with the code in line 90. No parameters are required.

```
 REM PROGRAM.071
10 X=USR(ADR("h[illegible]"))
20 ON X GOTO 40,50,60,70
30 ? "NEUTRAL":GOTO 10
40 ? "UP":GOTO 10
50 ? "DOWN":GOTO 10
60 ? "LEFT":GOTO 10
70 ? "RIGHT":GOTO 10
80 REM ** USE THE FOLLOWING MACHINE
   CODE FOR JOYSTICK 1.
90 X=USR(ADR("h[illegible]"))
```

PROGRAM.072 Joystick 0 or 1 - 9 Position Read

This program is very similar to the last program but now all 9 positions of the joystick will be read and translated into the new numbering system.

```
0 REM PROGRAM.072
10 X=USR(ADR("h[illegible]"))
20 ON X GOTO 40,50,60,70,80,90,100,110
30 ? "NEUTRAL":GOTO 10
40 ? "UP":GOTO 10
50 ? "DOWN":GOTO 10
60 ? "LEFT":GOTO 10
70 ? "RIGHT":GOTO 10
80 ? "UPPER LEFT":GOTO 10
90 ? "UPPER RIGHT":GOTO 10
100 ? "DOWN RIGHT":GOTO 10
110 ? "DOWN LEFT":GOTO 10
120 REM ** FOR JOYSTICK 1 USE THE
    FOLLOWING MACHINE ROUTINE.
130 X=USR(ADR("h[illegible]"))
```

PROGRAM.073 Any Joystick - 5 Position Read

This program is like PROGRAM.071 but it requires the number of the joystick (0-3) as the only parameter, line 10. The program below is set up to read Joystick 0.

P1 Which joystick, 0-3.

```
0 REM PROGRAM.073
10 X=USR(ADR("h[illegible]hh[illegible]JJJJ[illegible]
[illegible]"),0)
20 ON X GOTO 40,50,60,70
30 ? "NEUTRAL":GOTO 10
40 ? "UP":GOTO 10
50 ? "DOWN":GOTO 10
60 ? "LEFT":GOTO 10
70 ? "RIGHT":GOTO 10
```

PROGRAM.074 Any Joystick - 9 Position Read

This program is like PROGRAM.072 but it requires the number of the joystick (0-3) as the only parameter. The program below is set up to read Joystick 1.

P1 Which joystick, 0-3.

```
0 REM PROGRAM.074
10 DIM P$(121)
20 P$(1)="h[illegible]hh[illegible]JJJJ[illegible]
[illegible]"
30 P$(102)="[illegible]"
40 X=USR(ADR(P$),1)
50 ON X GOTO 70,80,90,100,110,120,130,
140
60 ? "NEUTRAL":GOTO 40
70 ? "UP":GOTO 40
80 ? "DOWN":GOTO 40
90 ? "LEFT":GOTO 40
100 ? "RIGHT":GOTO 40
110 ? "UPPER LEFT":GOTO 40
120 ? "UPPER RIGHT":GOTO 40
130 ? "DOWN RIGHT":GOTO 40
140 ? "DOWN LEFT":GOTO 40
```

PROGRAM.075 Joystick 0 & 1 - 9 Position Read

The program below reads all the positions of Joysticks 0 and 1 at the same time. The converted value of Joystick 0 is in the Return Variable and memory location 203. Its ON...GOTO statement is line 50. The value of Joystick 1 is in memory location 204 and its ON...GOTO statement is line 70. The sample program below will print the position of the two joysticks in two separate columns.

```
0 REM PROGRAM.075
10 DIM P$(115)
20 P$(1)="h[illegible]"
30 P$(102)="[illegible]JJJJ[illegible]"
40 X=USR(ADR(P$))
50 ON X GOTO 90,100,110,120,130,140,15
0,160
60 ? "NEUTRAL     ";
70 ON PEEK(204) GOTO 170,180,190,200,2
10,220,230,240
80 ? "           NEUTRAL":GOTO 40
90 ? "UP           ";:GOTO 70
100 ? "DOWN        ";:GOTO 70
110 ? "LEFT        ";:GOTO 70
120 ? "RIGHT       ";:GOTO 70
130 ? "UPPER LEFT  ";:GOTO 70
140 ? "UPPER RIGHT";:GOTO 70
150 ? "DOWN RIGHT  ";:GOTO 70
160 ? "DOWN LEFT   ";:GOTO 70
170 ? "          UP":GOTO 40
180 ? "          DOWN":GOTO 40
190 ? "          LEFT":GOTO 40
200 ? "          RIGHT":GOTO 40
210 ? "          UPPER LEFT":GOTO 40
220 ? "          UPPER RIGHT":GOTO 40
230 ? "          DOWN RIGHT":GOTO 40
240 ? "          DOWN LEFT":GOTO 40
```

PROGRAM.076 Joystick 2 & 3 - 9 Position Read

The program below is the mate to PROGRAM.075 for Joysticks 2 and 3. You can only use this routine on Atari 400 and 800s. Note the position value of Joystick 2 is in memory location 205 and the position value of Joystick 3 is in memory location 206. The Return Variable does not contain any useful value. No parameters are required.

```
0 REM PROGRAM.076
10 DIM P$(110)
20 P$(1)="h[illegible]"
30 P$(102)="[illegible]JJJJ[illegible]"
40 X=USR(ADR(P$))
50 ON PEEK(205) GOTO 90,100,110,120,13
0,140,150,160
60 ? "NEUTRAL      ";
70 ON PEEK(206) GOTO 170,180,190,200,2
10,220,230,240
80 ? "          NEUTRAL":GOTO 40
90 ? "UP           ";:GOTO 70
100 ? "DOWN        ";:GOTO 70
110 ? "LEFT        ";:GOTO 70
120 ? "RIGHT       ";:GOTO 70
130 ? "UPPER LEFT  ";:GOTO 70
140 ? "UPPER RIGHT";:GOTO 70
150 ? "DOWN RIGHT  ";:GOTO 70
160 ? "DOWN LEFT   ";:GOTO 70
170 ? "          UP":GOTO 40
180 ? "          DOWN":GOTO 40
190 ? "          LEFT":GOTO 40
200 ? "          RIGHT":GOTO 40
210 ? "          UPPER LEFT":GOTO 40
220 ? "          UPPER RIGHT":GOTO 40
230 ? "          DOWN RIGHT":GOTO 40
240 ? "          DOWN LEFT":GOTO 40
```

PROGRAM.077 Four Joystick Read - All Positions

This program will read all 4 joysticks and place their converted positions in memory locations 203, 204, 205, and 206 respectively. The numerical value in each joystick's position will be printed in separate columns by line 50 in the sample program. No parameters are required.

```
0 REM PROGRAM.077
10 DIM P$(135)
20 P$(1)="h[illegible])[illegible]3[illegible]@[illegible]M[illegible]
[illegible]N[illegible]A[illegible]
[illegible]"
30 P$(102)="[illegible]
[illegible]JJJJ[illegible]"
40 X=USR(ADR(P$))
50 FOR JOY=0 TO 3:? PEEK(203+JOY),:NEX
T JOY:? "↑":GOTO 40
60 REM 1=UP, 2=DOWN, 3=LEFT, 4=RIGHT,
   5=UPPER RIGHT, 6=UPPER LEFT,
   7=DOWN RIGHT, 8=DOWN LEFT.
```

Chapter 11
Player/Missile Graphics

Introduction

The subject of Player-Missile (P/M) Graphics almost deserves a book by itself. We will try to give some different machine routines to take the drudgery out of setting up and moving P/M. For more information on this subject, we recommend the book "Your Atari Computer" from Osborne/McGraw-Hill, or any of the fine articles on this subject in Analog or Antic magazines.

Player-Missiles are special graphic objects. They are independent of the background screen, and are designed for rapid movement. You can use up to four players and four missiles. Each of the four players can be as tall as a full screen and up to 8 bits (1 byte) wide. Both players and missiles can be displayed in single or double line resolution, and single, double, or quadruple horizontal width. In addition, each player can have it's own color and its priority with background objects and other players specified. Special collision registers can also detect the "collision" of one player with missiles, other players, and background objects.

Missiles are similar to players except they can only be 1 or 2 bits wide, and they must be the same color as their corresponding player. The four missiles can even be combined to form a fifth player!

To take advantage of all this power some set up work is required by the user. Although the following explanation may be above a beginner's level, you'll still learn quite a bit about P/M by working your way through the explanation and running the example programs.

P/M Memory

Player/Missile graphics require a special area of memory set aside for its use. The amount of memory depends on whether single or double resolution graphics are used. Single resolution graphics require 2048 bytes of memory, double resolution requires 1024 bytes of memory. This memory is partitioned automatically by the computer, as shown in Figure

1. In single resolution, each row of a player's shape will be displayed on one P/M line. For double resolution, each row of the player's shape will be displayed on two P/M lines, doubling the height of the player, but lessening its resolution. There are 192 P/M lines for a player or missile in single resolution, 96 in double resolution, regardless of the background GRAPHICS screen.

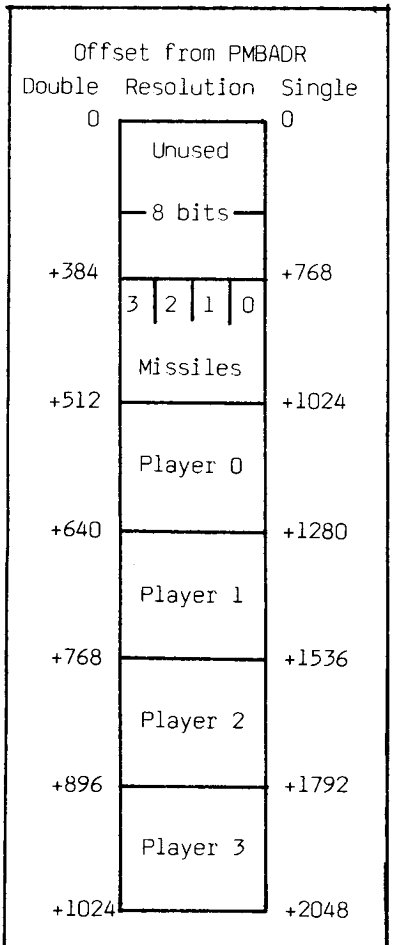


FIGURE 1: P/M Memory Layout

This special area of memory must be located in the computer where it will not be disturbed. Normally, the P/M memory can be located just below the background screen's Display List. For single resolution the P/M table must start at an address that is evenly divisible by 2048. For double resolution, it must be evenly divisible by 1024. The high byte of this address is called PMBASE for Player/Missile Base. The low byte is always zero.

Normally, PMBASE is fixed in a BASIC program by the number of pages (256 bytes) it must be setback from the top of memory, RAMTOP, which is determined by PEEK(106). For example, for a 48K Atari computer, a GRAPHICS 8 background screen, and single resolution P/M Graphics, a 40 page setback is required from RAMTOP (=160). The BASIC statement to define PMBASE would be:

```
PMBASE=PEEK(106)-40
```

The whole address would be:

```
PMBADR=(PEEK(106)-40)*256
```

Figure 2 gives the pages setback required for popular background (playfield) screens. The P/M table is setup in the computer with a simple POKE 54279,PMBASE. Single resolution graphics is selected by a POKE 559,62 and double resolution graphics by a POKE 559,46. To activate the players and missiles you must also do a POKE 53277,3.

Pages Setback From RAMTOP, (location 106)

Playfield	Resolution	
GRAPHICS	SINGLE	DOUBLE
0	16	8
1+16	16	8
2+16	16	8
7+16	32	28
8+16	40	36

FIGURE 2: Player Missile Starting Address

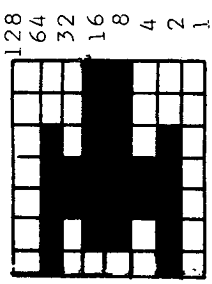
Defining The Player

To define a player's shape, start with some graph paper, mark off a column eight blocks wide and design your figure. Give each filled square on the grid a value of 1, and empty squares a 0 (see Figure 3). When you read straight across each line on the grid, you'll have an eight bit binary number. Convert the binary number to a decimal number with

PROGRAM.110, or by adding up the column values as shown in Figure 3. This series of decimal numbers define the player's shape when placed in the player's memory cell.

These numbers can be read into memory by placing them into DATA statements and using a FOR...NEXT loop, or by placing their character equivalents into a string and then copying it to the P/M memory area. In the example programs in this book, all P/M shapes will be placed in a string, because DATA statements are memory wasteful and take time to process. A string can be copied almost immediately to the P/M memory area with the programs in this book. The character corresponding to the ATASCII decimal number can be obtained from the Appendix.

Column Values



Binary Number	Decimal ATASCII Number
00011000	24
00011000	24
01011010	90
01111110	126
01111110	126
01011010	90
01000010	66

Character String = "┴┴Z◀◀ZB"

FIGURE 3: Defining A Player's Shape

The data's position within the memory cell for each player determines its vertical location on the screen. For example, take Player 0 in double resolution. Its shape data can lie between +512 and +640 bytes above PMBADR. If the shape data is placed near +512 bytes, the player will appear near the top of the screen. If the shape data is placed near +640 bytes, it will appear near the bottom. By moving the data within the memory cell the shape will travel up and down the screen.

Establishing a horizontal position and moving the player horizontally is a snap compared to vertical positioning and movement. Just POKE a number into the player's horizontal position register and the player will move there almost instantly (see Figure 4).

The horizontal position number can range from 0 to 227.

The left border of the screen is about 46, the right, 201. This positioning scheme allows the player to disappear off the screen on both sides. The equivalent vertical positioning (as an offset from the beginning memory location for each player) is 0-255 in single resolution with the borders around 31 and 217. For double, the values are 0-128 with the borders around 15 and 106.

P/M Width, Color, Priority, And Collision Detection

The overall width of the player can be changed with a single register. POKE a 0, 1, or 3 in the proper size register (see Figure 4) to get normal, double, or quadruple width.

A player can have its color independently selected through its color register as shown in Figure 4.

The priority of the player over background objects or other players can be chosen by the priority register, memory location 623. This is how a player can be made to duck behind a background object or appear in front of another player. Figure 4 describes this register.

One final aspect of P/M Graphics is the detection of "collisions". By reading (PEEKing) the collision registers (Figure 4) you can determine if a player has struck a background object or another player, or if a missile has struck a player or background object. The decimal number returned from one of the collision registers of FIGURE 4 is as follows:

No. of 2nd Object, Figure 4:	3	2	1	0
Resulting Decimal Number:	8	4	2	1

For example, if Player 1 collides with a playfield object made with color from color register 709 (#1), then PEEK(53253) will return a value of '2'. The playfield number of 0-3 corresponds to color registers 708, 709, 710, and 712.

The collision registers do not automatically reset themselves and you can not write to them. Consequently, register 53278 was set up to clear these registers. POKE 53278 with any number to clear all collision registers.

```
  559  (W) 62 for single, 46 for double line resolution
  623  (W) Sets player/playfield priorities
        1: All players priority over all playfields
        2: P0 & P1,then playfields, then P2 & P3
        4: All playfields priority over all players
        8: PF0 & PF1,then players,then PF2 & PF3
       16: Use 4 missiles as fifth player
  704  (W) Color of player/missile 0
  705  (W) Color of player/missile 1
  706  (W) Color of player/missile 2
  707  (W) Color of player/missile 3
53248  (W) Horizontal position player 0
       (R) Collision: Missile 0 to playfield
53249  (W) Horizontal position player 1
       (R) Collision: Missile 1 to playfield
53250  (W) Horizontal position player 2
       (R) Collision: Missile 2 to playfield
53251  (W) Horizontal position player 3
       (R) Collision: Missile 3 to playfield
53252  (W) Horizontal position missile 0
       (R) Collision: Player 0 to playfield
53253  (W) Horizontal position missile 1
       (R) Collision: Player 1 to playfield
53254  (W) Horizontal position missile 2
       (R) Collision: Player 2 to playfield
53255  (W) Horizontal position missile 3
       (R) Collision: Player 3 to playfield
53256  (W) Size player 0; 0, 1, or 3
       (R) Collision: Missile 0 to players
53257  (W) Size player 1; 0, 1, or 3
       (R) Collision: Missile 1 to players
53258  (W) Size player 2; 0, 1, or 3
       (R) Collision: Missile 2 to players
53259  (W) Size player 3; 0, 1, or 3
       (R) Collision: Missile 3 to players
53260  (W) Size of missiles.  See note
       (R) Collision: Player 0 to players
53261  (R) Collision: Player 1 to players
53262  (R) Collision: Player 2 to players
53263  (R) Collision: Player 3 to players
53277  (W) 3 to enable P/M Graphics, 0 to disable
53278  (W) Clear all collision registers
54279  (W) Put PMBASE here
```

(W) means write (POKE) and (R) means read (PEEK)

Note: For 53260 place a 0, 1, or 3 in the appropriate two bits for each missile for normal, double, or quadruple size. For example, 0 gives all four missiles normal size and 255 gives all quadruple size.

Important Player/Missile Memory Locations

FIGURE 4

PROGRAM.078 P/M In BASIC

This first listing of a Player/Missile program is mostly in BASIC to allow you to follow the progress in converting the program to machine language routines. Only the joystick read routine and the clearing of the player memory cell are in machine language because these routines have been presented earlier.

Line 10 DIMensions the string 'PLAYER$' which holds the shape information for the player and sets up a GRAPHICS 0 black background screen with a red border. The 'POKE 752,1:?' eliminates the cursor. Line 20 establishes the Player/Missile Base Address's high byte (PMBASE), sets up the P/M memory table, and finds the start of the first player, PLAY0, within the memory just setup. Line 30 is PROGRAM.010 which clears the special area of memory just setup since any stray data in that memory will show up on the screen. Line 40 chooses single line resolution and player priority over the background screen. Line 50 activates Player/Missile graphics.

Line 60 contains information about the player. X and Y determines the initial location on the screen for the player. POKE 704,20 gives the player a color and POKE 53256,0 selects normal width for the player.

Line 70 places the player's shape data, Figure 2, in the string PLAYER$ and inserts this information into the Player 0 memory cell. The player's vertical height, the number of characters in PLAYER$, is assigned to the variable SIZE0. Line 80 sets the initial horizontal position of Player 0 on the screen.

Line 90 is a joystick routine, PROGRAM.071, which works with line 100 to send the program to the proper move routine; up, down, left, or right. Lines 500 and 510 are the two vertical position routines. They simply move the shape data of the player higher or lower in the Player 0 memory cell, thereby, moving the player on the screen. The variable 'Y' keeps track of the vertical position on the screen with Y=0 as the start of the Player 0 memory cell, PLAY0. Usually, we refer to the value of 'Y' as an offset from PLAY0.

Lines 520 and 530 are the left and right BASIC move routines. The variable 'X' keeps track of the horizontal position of the player.

One special warning. If you move the player too far off screen the program may crash. For the programs in this book the 'Y' offset value can range from 2 to 253 in single resolution and 2 to 126 in double resolution. The horizontal position register 'X' can range from 2 to 225. Actually, the values can range from 0 to 255 for 'Y' (0-128 in double resolution) and 0 to 227 for 'X' but for reasons dealing with the way the player is moved and erased, use the tighter limits. In later programs we will show how to control these limits of travel to prevent the program from crashing. Run the program to get an idea of the speed of the vertical movement before moving on to the next programs.

```
0 REM PROGRAM.078
10 DIM PLAYER$(7):GRAPHICS 0:POKE 710,
0:POKE 712,66:POKE 752,1:?
20 PMBASE=PEEK(106)-16:POKE 54279,PMBA
SE:PLAY0=PMBASE*256+1024:REM SETUP P/M
   MEMORY AREA.
30 BB=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]hh[illegible]
[illegible]"),PLAY0,256,0):RE
M PROGRAM.010 ,CLEAR P/M AREA.
40 POKE 559,62:POKE 623,1
50 POKE 53277,3:REM ACTIVATE P/M
60 X=80:Y=70:POKE 704,20:POKE 53256,0:
REM INITIAL POSITION,COLOR,AND WIDTH
   OF PLAYER ZERO.
70 SIZE0=7:PLAYER$="[illegible]Z[illegible]ZB":FOR I=0 T
O 6:POKE PLAY0+I+Y,ASC(PLAYER$(I+1,I+1
)):NEXT I
80 POKE 53248,X
90 BB=USR(ADR("h[illegible]
[illegible]")):REM *
   PROGRAM.071, JOYSTICK ROUTINE.
100 ON BB GOTO 500,510,520,530
110 GOTO 90
500 FOR N=0 TO SIZE0:POKE PLAY0+Y-1+N,
PEEK(PLAY0+Y+N):NEXT N:Y=Y-1:GOTO 90
510 FOR N=SIZE0-1 TO -1 STEP -1:POKE P
LAY0+Y+N+1,PEEK(PLAY0+Y+N):NEXT N:Y=Y+
1:GOTO 90
520 X=X-1:POKE 53248,X:GOTO 90
530 X=X+1:POKE 53248,X:GOTO 90
```

PROGRAM.079 Vertical Movement Routine

This program introduces two machine routines in lines 500 and 510 for quicker and smoother vertical movement. Lines 10 through 110 are the same as in PROGRAM.078. Two parameters are required.

P1 Starting address of the player being moved plus the 'Y' vertical position offset.
P2 Vertical height of the player.

Lines 600 through 630 show the changes to make for double line resolution. You should make these changes to understand what double resolution looks like.

Placing other players on the screen is easy. Just specify their starting address, as in line 20, and place its shape data in the proper player memory cell. For example, Player 1 would be specified as PLAY1 = PMBASE*256+1280 (single resolution), and lines 60 through 80 would be repeated with characteristics for this player.

```
0 REM PROGRAM.079
10 DIM PLAYER$(7):GRAPHICS 0:POKE 710,
0:POKE 712,66:POKE 752,1:?
20 PMBASE=PEEK(106)-16:POKE 54279,PMBA
SE:PLAY0=PMBASE*256+1024
30 BB=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]hh[illegible]
[illegible]"),PLAY0,256,0):RE
M PROGRAM.010
40 POKE 559,62:POKE 623,1
50 POKE 53277,3
60 X=80:Y=70:POKE 704,20:POKE 53256,0
70 SIZE0=7:PLAYER$="┴┴Z◄◄ZB":FOR I=0 T
O 6:POKE PLAY0+I+Y,ASC(PLAYER$(I+1,I+1
)):NEXT I
80 POKE 53248,X
90 BB=USR(ADR("h[illegible]
[illegible]")):REM *
   PROGRAM.071
100 ON BB GOTO 500,510,520,530
110 GOTO 90
500 BB=USR(ADR("hh[illegible]h[illegible]hh[illegible]
[illegible]"),PLAY0+Y,SIZE0):Y=Y-1:GOTO 90
510 BB=USR(ADR("hh[illegible]h[illegible]hh[illegible]
"),PLAY0+Y,SIZE0):Y=Y+1:GOTO 90
520 X=X-1:POKE 53248,X:GOTO 90
530 X=X+1:POKE 53248,X:GOTO 90
600 REM **
610 REM FOR DOUBLE LINE RESOLUTION
    CHANGE LINE 620 TO LINE 20 AND
    CHANGE LINE 630 TO LINE 40.
620 PMBASE=PEEK(106)-8:POKE 54279,PMBA
SE:PLAY0=PMBASE*256+512
630 POKE 559,46:REM DOUBLE LINE
    RESOLUTION.
```

PROGRAM.080 2-Line Vertical Movement Routine

ENTER this program into the previous one for faster vertical movement. The player is moved 2 lines vertically each time one of the vertical machine routines is executed instead of one line as for the previous program. Note the vertical position offset 'Y' is changed by a value of plus or minus 2 to reflect this change.

```
0 REM PROGRAM.080
500 BB=USR(ADR("hh[illegible]h[illegible] hh[illegible]
[illegible]"),PLAY0+Y,SIZE0):Y=Y-2:GOTO 9
0
510 BB=USR(ADR("hh[illegible]h[illegible]hh[illegible]
[illegible]"),PLAY0+Y,SIZE0):Y=Y+2:GOTO 90
```

PROGRAM.081 P/M Address Setup Routine

In this example program we will introduce two machine routines, line 20 and 30, which do a lot of the setup work for P/M Graphics. The first routine in line 20 sets up the P/M memory areas, the P/M priorities, and the resolution. The Return Variable contains the Player/Missile Base Address, PMBADR. The machine routine in line 30 is a special P/M memory clearing routine which will only work after the line 20 routine.

The example program changes the background screen to a GRAPHICS 8 screen and selects double resolution just to show how it is done. The variable 'RES' is assigned to the Resolution selected.

The vertical move routines in lines 500 and 510 are the 2 byte move routines from PROGRAM.080.

The machine routines in lines 20 and 30 replaces the BASIC lines 20 throgh 50 in the previous programs. Three parameters are required for the machine routine in line 20.

P1 Pages setback from RAMTOP for the background screen selected (See FIGURE 2).

P2 Priority of Players, Missiles, and Background. See FIGURE 4, location 623.

P3 Single resolution = 1, Double resolution = 2.

```
0 REM PROGRAM.081
10 DIM PLAYER$(7):GRAPHICS 24:POKE 710
,0:POKE 712,66:RES=2
20 PMBADR=USR(ADR("hhh[illegible]j8[illegible]hh[illegible]o
[illegible]>hh[illegible].[illegible]/[illegible].[illegible]
[illegible]e[illegible]"),40,1,RES)
30 Z=USR(ADR("h[illegible]
[illegible]")):REM CLEARS P/M AREA.  MUST
   BE USED AFTER LINE 20.
40 PLAY0=PMBADR+512
60 X=80:Y=70:POKE 704,20:POKE 53256,0
70 SIZE0=7:PLAYER$="┴┴Z◀◀ZB":FOR I=0 T
O 6:POKE PLAY0+I+Y,ASC(PLAYER$(I+1,I+1
)):NEXT I
80 POKE 53248,X
90 BB=USR(ADR("h[illegible]
[illegible]")):REM *
   PROGRAM.071
100 ON BB GOTO 500,510,520,530
110 GOTO 90
500 BB=USR(ADR("hh[illegible]h[illegible]hh[illegible]
[illegible]"),PLAY0+Y,SIZE0):Y=Y-2:GOTO 9
0
510 BB=USR(ADR("hh[illegible]h[illegible]hh[illegible]
[illegible]"),PLAY0+Y,SIZE0):Y=Y+2:GOTO 90
520 X=X-1:POKE 53248,X:GOTO 90
530 X=X+1:POKE 53248,X:GOTO 90
```

PROGRAM.082 Player Setup Routine

The next program moves the player's shape data into the proper memory location using a machine routine. Lines 70 and 80 is the machine routine that replaces the BASIC lines 40 through 80 in the previous program. Eight parameters are required which all describe the player of interest. Note that lines 520 and 530 have been changed slightly to show how the horizontal movement of the player is prevented from going off screen and crashing the program.

P1 The Player/Missile Base Address, PMBADR.
P2 Player number, 0-3.
P3 Address of the player shape data.
P4 Initial horizontal position of the player.
P5 Overall width of the player, 0, 1, or 3.
P6 Single resolution=1, Double resolution=2.
P7 Initial vertical position offset of the player, 'Y'.
P8 Vertical height of the player.

```
0 REM PROGRAM.082
10 DIM PLAYER$(7),P$(92):GRAPHICS 24:P
OKE 710,0:POKE 712,66:RES=1
20 PMBADR=USR(ADR("hhh[illegible]"),40,3,RES)
30 Z=USR(ADR("h[illegible]")):REM CLEARS P/M AREA.  MUST
   BE USED AFTER LINE 20.
40 X=80:Y=70:SIZE0=7:POKE 704,20:WIDTH
=1
60 PLAYER$="[illegible]Z[illegible]ZB"
70 P$="hh[illegible]"
80 PLAY0=USR(ADR(P$),PMBADR,0,ADR(PLAY
ER$),X,WIDTH,RES,Y,SIZE0)
90 BB=USR(ADR("h[illegible]")):REM *
   PROGRAM.071
100 ON BB GOTO 500,510,520,530
110 GOTO 90
500 BB=USR(ADR("hh[illegible]"),PLAY0+Y,SIZE0):Y=Y-2:GOTO 9
0
510 BB=USR(ADR("hh[illegible]"),PLAY0+Y,SIZE0):Y=Y+2:GOTO 90
520 X=X-1:POKE 53248,X:IF X<46 THEN X=
46
525 GOTO 90
530 X=X+1:POKE 53248,X:IF X>201 THEN X
=201
540 GOTO 90
```

PROGRAM.083 Vertical Movement With Wrap Around.

The vertical movement routines are in lines 500 and 510. The string PU$ contains the 'up' routine and the string PD$ contains the 'down' routine. These routines allow the choice of stopping the vertical movement at any selected location (see the last paragraph of PROGRAM.078 screen limits), or allowing the player to wrap around the screen. In addition, these routines keep track of the 'Y' location of the player in the Return Variable. Six parameters are required for each routine.

The horizontal BASIC move routines show how the player can wrap around the screen. For a faster version of these two routines, see lines 83 and 84 in PROGRAM.084.

Up Move Routine, Line 500.

- P1 Base address of the player being moved.
- P2 Present vertical position of the player, 'Y'.
- P3 Vertical height of the player.
- P4 Upper movement limit of the player.
- P5 Lower movement limit of the player.
- P6 0 for stopping at limits, 1 for wrapping around screen.

Down Move Routine, Line 510.

- P1 Base address of the player being moved.
- P2 Present vertical position of the player, 'Y'.
- P3 Upper movement limit of the player.
- P4 Vertical height of the player.
- P5 Lower movement limit of the player.
- P6 0 for stopping at limits, 1 for wrapping around screen.

```
0 REM PROGRAM.083
10 DIM PLAYER$(7),P$(92),PU$(85),PD$(8
4):GRAPHICS 24:POKE 710,0:POKE 712,66:
RES=1
20 PMBADR=USR(ADR("hhh[illegible]j8[illegible]hh[illegible]o
[illegible]>hh[illegible] [illegible].[illegible]/ [illegible].[illegible]
[illegible]e[illegible]"),40,1,RES)
30 Z=USR(ADR("h[illegible]
[illegible]")):REM CLEARS P/M AREA.  MUST
   BE USED AFTER LINE 20.
40 X=80:Y=70:UL=31:DL=220:SIZE0=7:POKE
 704,20:WIDTH=0
60 PLAYER$="[illegible]Z[illegible]ZB"
70 P$="hh[illegible]h[illegible]hh[illegible]h[illegible]h[illegible]hh[illegible]hh[illegible]
[illegible]hh[illegible]
hh[illegible]hh[illegible]"
80 PLAY0=USR(ADR(P$),PMBADR,0,ADR(PLAY
ER$),X,WIDTH,RES,Y,SIZE0)
83 PU$="h[illegible]h[illegible]h[illegible]hh[illegible]hh[illegible]hh[illegible]
[illegible]hhh[illegible]hh[illegible]hh[illegible]8[illegible]
[illegible]"
86 PD$="h[illegible]h[illegible]h[illegible]hh[illegible]hh[illegible]hh[illegible]e[illegible]
[illegible]hhh[illegible]hh[illegible]hh[illegible]
[illegible]"
90 BB=USR(ADR("h[illegible]) [illegible]) [illegible]
[illegible]")):REM *
   PROGRAM.071
100 ON BB GOTO 500,510,520,530
110 GOTO 90
500 Y=USR(ADR(PU$),PLAY0,Y,SIZE0,UL,DL
,0):GOTO 90
510 Y=USR(ADR(PD$),PLAY0,Y,DL,SIZE0,UL
,0):GOTO 90
520 X=X-1:POKE 53248,X:IF X<46 THEN X=
201
525 GOTO 90
530 X=X+1:POKE 53248,X:IF X>201 THEN X
=46
540 GOTO 90
```

PROGRAM.084 Horizontal Movement With Wrap Around

This program introduces two horizontal move routines in machine language, which allow stopping at selected limits or wrapping around the screen. The two routines in lines 520 and 530 require six parameters each. The parameters are the same, just in a different order. Note that the Return Variable must be the value of the present horizontal position value, 'X' in this example.

The vertical move routines in lines 83 and 86 are faster versions of the routines in PROGRAM.083.

Left Move Routine, Line 520

- P1 Which player, 0-3.
- P2 Left move limit.
- P3 Horizontal amount to jump each time the machine routine is called.
- P4 Present horizontal position of the player, 'X' in our example.
- P5 0 to stop at limits, 1 to wrap around screen.
- P6 Right move limit.

Right Move Routine, Line 530

- P1 Which player, 0-3.
- P2 Right move limit.
- P3 Horizontal amount to jump each time the machine routine is called.
- P4 Present horizontal position of the player, 'X' in our example.
- P5 0 to stop at limits, 1 to wrap around screen.
- P6 Left move limit.

```
0 REM PROGRAM.084
10 DIM PLAYER$(7),P$(92),PU$(94),PD$(8
9):GRAPHICS 24:POKE 710,0:POKE 712,66:
RES=1
20 PMBADR=USR(ADR("hhh[illegible]j8[illegible]hh[illegible]o
[illegible]>hh[illegible].[illegible]/[illegible].[illegible]
[illegible]e[illegible]"),40,1,RES)
30 Z=USR(ADR("h[illegible]
[illegible]")):REM CLEARS P/M AREA.  MUST
   BE USED AFTER LINE 20.
40 X=80:Y=70:DL=224:UL=31:LL=46:RL=201
:SIZE0=7:POKE 704,20:WIDTH=0
60 PLAYER$="[illegible]Z[illegible]ZB"
70 P$="hh[illegible]h[illegible]hh[illegible]h[illegible]h[illegible]hh[illegible]hh[illegible]
[illegible]hh[illegible]
hh[illegible]hh[illegible]"
80 PLAY0=USR(ADR(P$),PMBADR,0,ADR(PLAY
ER$),X,WIDTH,RES,Y,SIZE0)
83 PU$="h[illegible]h[illegible]h[illegible]hh[illegible]hh[illegible]hh[illegible]
[illegible]hhhh[illegible]hh[illegible]hh[illegible]8[illegible]
[illegible]"
86 PD$="h[illegible]h[illegible]h[illegible]hh[illegible]hh[illegible]hh[illegible]e[illegible]
[illegible]hhhh[illegible]hh[illegible]hh[illegible]
[illegible]"
90 BB=USR(ADR("h[illegible])[illegible])[illegible]
[illegible]")):REM *
   PROGRAM.071
100 ON BB GOTO 500,510,520,530
110 GOTO 90
500 Y=USR(ADR(PU$),PLAY0,Y,SIZE0,UL,DL
,1):GOTO 90
510 Y=USR(ADR(PD$),PLAY0,Y,DL,SIZE0,UL
,1):GOTO 90
520 X=USR(ADR("h[illegible]hh[illegible]hh[illegible]hh[illegible]hh[illegible]8[illegible]
[illegible]hhhh[illegible]hh[illegible]hh[illegible]"),0,LL,3,X,
0,RL):GOTO 90
530 X=USR(ADR("h[illegible]hh[illegible]hh[illegible]hh[illegible]hh[illegible]e[illegible]
[illegible]hhhh[illegible]hh[illegible]hh[illegible]"),0,RL,3,X,
0,LL):GOTO 90
```

PROGRAM.085 Missile Mover

Like players, missiles can be defined vertically the whole length of the screen but only up to 2 bits wide. All four missiles are placed in one memory cell as shown in Figure 1. Missile 0 is described by the first two bits in each byte of the memory cell, Missile 1 by the next two bits, etc..

To move a missile in the horizontal axis you use the horizontal position register for the missile just like in moving a player. Moving a missile in the vertical position is a little tougher than a player. When a missile is moved the other bits in the byte must not be moved and the destination location must not have the other bits disturbed.

Line 70 contains the correct routine to setup a missile which is different than the one used to set up a player. The parameters, however, are the same as for the player routine except all the needed information is for the missile. The missile's shape is defined by line 60 as two lines high and two bits wide. Lines 500 and 510 contain the missile move routines. Three parameters are required for these routines.

- P1 Base address of the missile memory cell plus the present vertical position offset value.
- P2 Vertical size of the missile.
- P3 Which missile, 0-3.

```
0 REM PROGRAM.85
10 DIM MISSILE$(2),P$(87):GRAPHICS 24:
POKE 710,0:POKE 712,66:RES=1
20 PMBADR=USR(ADR("hhh[illegible]j8[illegible]hh[illegible]o
[illegible]>hh[illegible].[illegible]/ [illegible].[illegible]h[illegible]
[illegible]e[illegible]"),40,1,RES)
30 Z=USR(ADR("h[illegible]
[illegible]")):REM CLEARS P/M AREA.  MUST
   BE USED AFTER LINE 20.
40 X=80:Y=70:SIZE0=2:POKE 704,66:WIDTH
=0
60 MISSILE$="[illegible]"
70 P$="hh[illegible]h[illegible]i[illegible]hh[illegible]0[illegible]h[illegible]
h[illegible]hh[illegible]hh[illegible]hh[illegible]hh[illegible]hh[illegible]
[illegible]/[illegible]"
80 MIS0=USR(ADR(P$),PMBADR,0,ADR(MISSI
LE$),X,WIDTH,RES,Y,SIZE0)
90 BB=USR(ADR("h[illegible])[illegible]
[illegible]")):REM *
   PROGRAM.071, JOYSTICK ROUTINE.
100 ON BB GOTO 500,510,520,530
110 GOTO 90
500 BB=USR(ADR("hh[illegible]h[illegible]hh[illegible]hh[illegible]0[illegible]
[illegible]I[illegible]1[illegible]1[illegible]h[illegible]"),MIS
0+Y,SIZE0,0):Y=Y-1:GOTO 90
510 BB=USR(ADR("hh[illegible]h[illegible]hh[illegible]hh[illegible]0[illegible]
[illegible]I[illegible]1[illegible]1[illegible]h[illegible]"),MIS0+Y
,SIZE0,0):Y=Y+1:GOTO 90
520 X=X-1:POKE 53252,X:GOTO 90
530 X=X+1:POKE 53252,X:GOTO 90
590 REM *
595 REM USE THE BELOW ROUTINES FOR
    LINES 500 AND 510 FOR A FASTER
    MOVE ROUTINE (2 BYTE MOVE).
600 BB=USR(ADR("hh[illegible]h[illegible]hh[illegible]hh[illegible]0
[illegible]I[illegible]1[illegible]1[illegible]h[illegible]"
),MIS0+Y,SIZE0,0):Y=Y-2:GOTO 90
610 BB=USR(ADR("hh[illegible]h[illegible]hh[illegible]hh[illegible]0[illegible]
[illegible]I[illegible]1[illegible]1[illegible]h[illegible]"),M
IS0+Y,SIZE0,0):Y=Y+2:GOTO 90
```

PROGRAM.086 Clear Player/Missiles

Now that you have all those P/M running around the screen you need some way to turn them off without using the 'RESET' key. Just jump to the machine routine below in your BASIC program to turn off the P/M Graphics. The routine clears all the P/M memory area, starting at the missile memory cell, deactivates P/M, and POKEs 559 with 34, the default value.

P1 P/M Base Address, PMBADR.

```
0 REM PROGRAM.086
10 BB=USR(ADR("hh[illegible]h[illegible]/ [illegible].[illegible]h[illegible]
[illegible]
[illegible]![illegible]/ [illegible]") ,PMBADR)
```

Chapter 12
Player/Missiles in Vertical Blank Interrupt

You may have noticed that the movement of the players in the previous examples is somewhat jerky. The reason is the players are being moved while the screen is being drawn. The screen is redrawn sixty times a second. When the scanning beam reaches the bottom right of the screen it turns off and moves to the upper left of the screen to begin again. During this screen off time period (The Vertical Blank Interrupt, VBI) the Atari computer does a little house cleaning, but does have time to execute a machine program. If we can move our player to during this time period, the jerky movement can be eliminated.

All the previous routines for setting up and defining a player can be used with the next routines. The VBI routine must be installed in a known area of memory. If you look at Figure 4 there is an area of P/M memory that is not used. This is an ideal area to place the VBI routines. The joystick routines are also placed in these routines, since direction of movement must also be sensed in the VBI time period.

In the following examples, we'll also do some collision detection, just to show how its done.

<u>PROGRAM.087</u> <u>One</u> <u>Player</u>

This program will move one player in the VBI. Setting up the P/M memory, defining the player, and installing the player in memory uses the same procedures as in the previous non VBI P/M routines. Lines 10 to 80 are nearly the same as the lines in PROGRAM.082 except we'll set up a GRAPHICS 0 background screen. Lines 90 to 100 contain the VBI joystick and player movement machine code. This code is installed in the unused P/M memory area by the machine routine in line 110 (Actually installed at PMBADR+3), which also does some other work, and activates the VBI routine. It requires six parameters.

P1 P/M Base Address, PMBADR.
P2 Vertical height of player.
P3 Initial horizontal position of the player.
P4 Initial vertical position offset of player.
P5 Address of the VBI code less 3.
P6 Length of the VBI code.

Lines 130 to 160 shows a simple collision detection routine. Line 130 places some inverse spaces on the background screen as objects to impact. Line 140 turns on a sound if the player hits these objects and line 150 turns the sound off when the objects are not touched. Line 160 constantly resets all collision registers for another detection. Refer to the appropriate register selections in FIGURE 4 for further information.

```
0 REM PROGRAM.087
10 DIM PLAYER$(7),P$(131):GRAPHICS 0:POKE 710,0:POKE 712,66:POKE 752,1:? :RES=1
20 PMBADR=USR(ADR("[illegible]"),16,1,RES)
30 Z=USR(ADR("[illegible]")):REM CLEARS P/M AREA.  MUST
   BE USED AFTER LINE 20.
40 X=80:Y=70:SIZE0=7:POKE 704,20:WIDTH=0
60 PLAYER$="[illegible]"
70 P$="[illegible]"
80 PLAY0=USR(ADR(P$),PMBADR,0,ADR(PLAYER$),X,WIDTH,RES,Y,SIZE0)
90 P$(1)="[illegible]"
100 P$(102)="[illegible]"
110 BB=USR(ADR("[illegible]"),PMBADR,SIZE0,X,Y,ADR(P$)-3,LEN(P$))
120 REM * JOYSTICK & PLAYER MOVE
    ROUTINES HAVE BEEN INSTALLED.  NOW
    LOOK FOR COLLISIONS.
130 POSITION 8,10:? "████          ████
    ████"
140 IF PEEK(53252)<>0 THEN SOUND 0,80,10,10
150 IF PEEK(53252)=0 THEN SOUND 0,0,0,0
160 POKE 53278,0:GOTO 140
```

PROGRAM.088 Two Players

The next program shows the set up and movement of two players. The second player is set up with its own defining characteristics in line 50. We'll use the same shape for the second player as the for the first player. The same machine routine in line 60 is used to set up this second player, same as the first player. Line 90 just makes another USR call to the same machine routine using the second player data.

Lines 100 and 110 contains the entire VBI code and is installed into the unused P/M memory area by the machine routine in line 130. This routine requires 9 parameters, three more than the one player routine to accommodate the second player.

P1 P/M Base Memory Address, PMBADR.
P2 Vertical height of player zero.
P3 Initial horizontal position of player zero.
P4 Initial vertical position offset of player zero.
P5 Vertical height of player one.
P6 Initial horizontal position of player one.
P7 Initial vertical position offset of player one.
P8 Address of the VBI routine less 6 bytes.
P9 Length of the VBI routine.

Lines 150 to 170 shows player to player collision detection. See the previous program for an explanation of this routine.

```
0 REM PROGRAM.088
10 DIM PLAYER$(7),P$(92),VBI$(166):GRA
PHICS 0:POKE 710,0:POKE 712,66:POKE 75
2,1:? :RES=1
20 PMBADR=USR(ADR("hhh[illegible]j8[illegible]hh[illegible]o
[illegible]>hh[illegible] [illegible].[illegible]/ [illegible]
[illegible]e[illegible]"),16,1,RES)
30 Z=USR(ADR("h[illegible]
[illegible]")):REM CLEARS P/M AREA.  MUST
   BE USED AFTER LINE 20.
40 X=80:Y=70:SIZE0=7:POKE 704,20:WIDTH
=0
50 X1=110:Y1=120:SIZE1=7:POKE 705,33:W
IDTH1=0
60 P$="hh[illegible]h[illegible]hh[illegible]h[illegible]h[illegible]hh[illegible]hh[illegible]
[illegible]hh[illegible]
hh[illegible]hh[illegible]"
70 PLAYER$="[illegible]Z[illegible]ZB"
80 PLAY0=USR(ADR(P$),PMBADR,0,ADR(PLAY
ER$),X,WIDTH,RES,Y,SIZE0)
90 PLAY1=USR(ADR(P$),PMBADR,1,ADR(PLAY
ER$),X1,WIDTH1,RES,Y1,SIZE1)
100 VBI$(1)="[illegible]i[illegible]x [illegible]y [illegible]
[illegible]
8[illegible]Lb[illegible]
"
110 VBI$(102)="[illegible]i [illegible]e[illegible]
[illegible]0[illegible]i [illegible]8[illegible]
[illegible]"
120 P$="hh[illegible]h[illegible] [illegible]hh[illegible]hh[illegible]hh[illegible]h[illegible]
[illegible]h[illegible]hh[illegible]\[illegible]"
130 BB=USR(ADR(P$),PMBADR,SIZE0,X,Y,SI
ZE1,X1,Y1,ADR(VBI$)-6,LEN(VBI$))
140 REM JOYSTICK/PLAYER MOVE ROUTINES
    HAVE BEEN INSTALLED.  NOW CHECK
    FOR PLAYER-PLAYER COLLISIONS.
150 IF PEEK(53260)<>0 THEN SOUND 0,80,
10,10
160 IF PEEK(53260)=0 THEN SOUND 0,0,0,
0
170 POKE 53278,0:GOTO 150
```

PROGRAM.089:One Player With Wrap Around

The last two routines will bomb if you move the player too far beyond the horizontal or vertical limits (See the last paragraph in PROGRAM.078 for the limits). Notice that there is no way to keep track of the players position in BASIC, as was the case in the non-VBI routines. Use the next routine to prevent the player from going out of bounds. It allows the player to stop or wrap around the screen at selected positions.

This program is similiar to PROGRAM.087 up to line 80, except for line 40, which contains the right screen limit (RL), the left screen limit (LL), the upper screen limit (UL), and the down screen limit (DL). Lines 90 to 110 contain the VBI code which is installed into the P/M unused memory area by PROGRAM.001 in line 120. The three parameters required for line 120 are:

P1 Address of the VBI code.
P2 P/M Base Address, PMBADR.
P3 Length of VBI routine.

Since we now have many screen limit numbers to keep track of we are going to place them in Page 6. Consequently, you must not use memory areas 1536 to 1546 in your BASIC program when the P/M are activated.

Line 130 contains the VBI set up code which is activated in line 140. This routine requires 10 parameters.

P1 P/M Base Address, PMBADR.
P2 Left screen limit.
P3 Right screen limit.
P4 Upper screen limit.
P5 Down screen limit.
P6 0 for stop at limits, 1 for wrap around screen.
P7 Vertical size of player.
P8 Initial horizontal position of player.
P9 Initial vertical position offset of player.
P10 Speed of horizontal movement, 1 being the slowest.

When you run the program you will notice that the player appears and can be moved even though the BASIC program ends after line 140 with the familiar 'READY' prompt. Remember that our P/M routine is not tied into the BASIC program, but occurs during the VBI interval. The BASIC program can go on doing its own thing as long as it doesn't mess up the P/M memory area or introduce another VBI routine.

```
0 REM PROGRAM.089
10 DIM PLAYER$(7),P$(254):GRAPHICS 0:P
OKE 710,0:POKE 712,66:POKE 752,1:? :RE
S=1
20 PMBADR=USR(ADR("hhh[illegible]"),16,1,RES)
30 Z=USR(ADR("h[illegible]")):REM CLEARS P/M AREA.  MUST
   BE USED AFTER LINE 20.
40 X=70:Y=100:LL=46:RL=201:UL=31:DL=22
0:POKE 704,20:WIDTH=0:SIZE0=7
50 X1=110:Y1=120:SIZE1=7:POKE 705,33:W
IDTH1=0
60 PLAYER$="[illegible]Z[illegible]ZB"
70 P$="hh[illegible]"
80 PLAY0=USR(ADR(P$),PMBADR,0,ADR(PLAY
ER$),X,WIDTH,RES,Y,SIZE0)
90 P$(1)="[illegible]"
100 P$(102)="[illegible]"
110 P$(200)="[illegible]"
120 BB=USR(ADR("hh[illegible]"),ADR(P$),PMBADR,LEN(P$)):REM *
    PROGRAM.001
130 P$="hh[illegible]"
140 BB=USR(ADR(P$),PMBADR,LL,RL,UL,DL,
1,SIZE0,X,Y,3)
```

PROGRAM.090 One Player, 2 Line Vertical Movement

This program is identical to PROGRAM.089 except it contains a faster vertical movement routine (2 byte move) in the VBI code of lines 90 to 110. Line 120 is from line 60 of PROGRAM.001, since more than 256 bytes are being moved.

```
0 REM PROGRAM.090
10 DIM PLAYER$(7),P$(270):GRAPHICS 0:P
OKE 710,0:POKE 712,66:POKE 752,1:? :RE
S=1
20 PMBADR=USR(ADR("hhh[illegible]j8[illegible]hh[illegible]o
[illegible]>hh[illegible]
[illegible]"),16,1,RES)
30 Z=USR(ADR("h[illegible]
[illegible]")):REM CLEARS P/M AREA.  MUST
   BE USED AFTER LINE 20.
40 X=70:Y=100:LL=46:RL=201:UL=31:DL=22
0:POKE 704,20:WIDTH=0:SIZE0=7
60 PLAYER$="[illegible]Z[illegible]ZB"
70 P$="hh[illegible]h[illegible]hh[illegible]
[illegible]hh[illegible]
hh[illegible]hh[illegible]"
80 PLAY0=USR(ADR(P$),PMBADR,0,ADR(PLAY
ER$),X,WIDTH,RES,Y,SIZE0)
90 P$(1)="[illegible]i[illegible]x[illegible]
[illegible]MLb[illegible]
[illegible]Lb[illegible]Z[illegible]"
100 P$(102)="m[illegible]Lb[illegible]
[illegible]Lb[illegible]
[illegible]"
110 P$(200)="[illegible]
[illegible]Lb[illegible]
[illegible]Lb[illegible]"
120 BB=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]h[illegible]h[illegible]
[illegible]"),ADR(P$)
,PMBADR,LEN(P$)):REM PROGRAM.001
130 P$="hh[illegible]h[illegible]hh[illegible]hh[illegible]hh[illegible]hh[illegible]
/hh[illegible]//hh[illegible]/hh[illegible]/hh[illegible]/hh[illegible]\[illegible]"
140 BB=USR(ADR(P$),PMBADR,LL,RL,UL,DL,
1,SIZE0,X,Y,3)
```

PROGRAM.091 Two Players With Wrap Around.

This program is the two player version of PROGRAM.088 with stop or wrap limits. Page 6 memory from 1536 to 1561 is used. A grand total of 17 parameters are used for the USR call in line 160.

P1 P/M Base Memory Address, PMBADR
P2 0 for stop at limits, 1 for wrap around screen.
P3 Speed of horizontal movement.
P4 Player zero left screen limit.
P5 Player zero right screen limit.
P6 Player zero upper screen limit.
P7 Player zero down screen limit.
P8 Player zero vertical size.
P9 Player zero Initial horizontal position.
P10 Player zero initial vertical position offset value.
P11 to P17 Player 1 equivalents of P4 to P10.

```
0 REM PROGRAM.091
10 DIM PLAYER$(7),P$(333):GRAPHICS 0:P
OKE 710,0:POKE 712,66:POKE 752,1:? :RE
S=1
20 PMBADR=USR(ADR("hhh[illegible]j8[illegible]hh[illegible]o
[illegible]>hh[illegible] [illegible].[illegible] [illegible].[illegible]
[illegible]e[illegible]"),16,1,RES)
30 Z=USR(ADR("h[illegible]
[illegible]")):REM CLEARS P/M AREA.  MUST
   BE USED AFTER LINE 20.
40 X=70:Y=100:LL=46:RL=201:UL=31:DL=22
0:POKE 704,20:SIZE0=7
50 X1=110:Y1=120:LL1=46:RL1=201:UL1=31
:DL1=220:POKE 705,33:SIZE1=7
60 P$="hh[illegible]hh[illegible]hh[illegible]hh[illegible]
[illegible]hh[illegible]
hh[illegible]hh[illegible]"
70 PLAYER$="[illegible]Z[illegible]ZB"
80 PLAY0=USR(ADR(P$),PMBADR,0,ADR(PLAY
ER$),X,WIDTH,RES,Y,SIZE0)
90 PLAY1=USR(ADR(P$),PMBADR,1,ADR(PLAY
ER$),X1,WIDTH1,RES,Y1,SIZE1)
100 P$(1)="[illegible]i[illegible]x [illegible]
[illegible]![illegible]y [illegible]
[illegible]B[illegible]3[illegible]/"
110 P$(102)="[illegible]0=[illegible]j[illegible]
[illegible]R[illegible]m[illegible]0[illegible]d[illegible]5[illegible]
[illegible]
"
120 P$(203)="[illegible]Lb[illegible]P[illegible]
[illegible]
[illegible]0[illegible]
"
130 P$(304)="[illegible]
[illegible]0[illegible]"
140 BB=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]h[illegible]h[illegible]
[illegible]"),ADR(P$)
,PMBADR,LEN(P$))
150 P$="h[illegible]h[illegible]h[illegible]hh[illegible]hh[illegible]hh[illegible]
/hh[illegible]/hh[illegible]//hh[illegible]/hh[illegible]/hh[illegible]/hh[illegible]
[illegible]"
160 BB=USR(ADR(P$),PMBADR,1,3,LL,RL,UL
,DL,SIZE0,X,Y,LL1,RL1,UL1,DL1,SIZE1,X1
,Y1)
```

PROGRAM.092 Clear P/M In VBI

When you are ready to clear the P/M from the screen and go about some other business in your BASIC program use the next routine. Note that any information in the unused area of the Player-Missile memory or Page 6 is not erased. Only the memory starting with the missile memory cell through the last player is erased. The routine also deactivates the necessary P/M registers. There are two different routines depending on the type of computer you have.

P1 P/M Base Address, PMBADR.

```
0 REM PROGRAM.092
10 REM * FOR ATARI 400/800 COMPUTER.
20 BB=USR(ADR("hh[illegible]h[illegible]/ [illegible].[illegible]h[illegible]
[illegible]
[illegible]![illegible]/ [illegible]h[illegible] \[illegible]"),PMBADR)
30 REM
40 REM * FOR ATARI XL/XE COMPUTER.
50 BB=USR(ADR("hh[illegible]h[illegible]/ [illegible].[illegible]h[illegible]
[illegible]
[illegible]![illegible]/ [illegible] \[illegible]"),PMBADR)
```

Chapter 13
Miscellaneous

PROGRAM.093 Random Non-Repeating Numbers

The following short program will generate up to 256 random numbers that do not repeat themselves. These numbers will range from 0 to 255 in value.

The variable 'SIZE' in line 10 is the number of non-repeating numbers desired. Line 20 generates numbers from 0 to one less than SIZE in numerical order. These numbers are placed in the string S$. The second parameter is the starting address of the second string or memory area where the non-repeating random numbers will be generated from the S$ string. The third parameter is the number of random numbers to generate.

P1 Starting address to generate first set of numbers from 0 to length of numbers desired.
P2 Starting address of non-repeating random numbers.
P3 Number of random numbers desired.

Lines 30 to 40 take the numbers out of the S$ string and randomizes them into the second memory area starting at 1536. Another string could also be used. Note that the USR call in line 40 uses some BASIC programing and must be written as is. It also must come after line 20 which sets up line 40 with some needed information. Line 50 looks at the generated random numbers to verify the numbers are non-repeating.

For this sample program we will randomize 52 numbers which can be assumed to be 52 playing cards. The single parameter for the second machine routine in line 40 is:

P1 INT(N*RND(0)), where N is the loop counter from line 30.

```
0 REM PROGRAM.093
10 SIZE=52:DIM S$(SIZE)
20 X=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]hh[illegible]
[illegible]"),ADR(S$),1536,SIZE)
30 FOR N=SIZE TO 1 STEP -1
40 X=USR(ADR("hhh[illegible]
[illegible]"),INT(N*RND(0))):NEXT N
50 FOR I=0 TO SIZE-1:? I,PEEK(1536+I):
NEXT I
```

PROGRAM.094 Atari Rainbow

The following program produces the famous ATARI rainbow effect. This effect can be done in GRAPHICS 0, 1, or 2 screens. For GRAPHICS 1 or 2 screens either the letters or the background screen can be 'rainbowed'. Line 10 sets up a GRAPHICS 2 screen and prints 'MACHINE LANGUAGE' on the screen in four different colors. The four different colors are assigned to color registers 708 to 711. The background screen color is register 712. Registers and colors are assigned as follows:

	GRAPHICS 0	GRAPHICS 1&2
Upper Case	709	708
Lower Case	709	709
Inverse Upper Case	709	710
Inverse Lower Case	709	711
Background	710	712
Border	712	712

Line 20 sets up a loop to cycle the rainbow through each of the color registers to show the full effect. The machine routine in line 40 or 70 needs three parameters.

P1 Time duration of Rainbow effect.
P2 Width of individual color lines, 1 is widest.
P3 Code number, 0-4, for color registers 708 to 712, respectively, to 'rainbow'.

```
0 REM PROGRAM.094
10 GRAPHICS 2+16:POSITION 6,2:? #6;"Ma
██I█e":POSITION 5,7:? #6;"l██G█u█E!"
20 FOR C=0 TO 4
30 X=USR(ADR("hhh██hh██hh██ █▼▼●█<┴e██
▶██ █ ██●██████♦"),4,1,C)
40 NEXT C
50 REM *
60 REM USE FOLLOWING ROUTINE FOR COLOR
   DENSITY OPPOSITE ABOVE ROUTINE.
70 X=USR(ADR("hhh██hh██hh██H█▼▼●█<8██
▶██ █ ██●██████♦"),4,1,C)
80 REM FIRST PARAMETER IS DELAY TIME,
   2ND IS LINE WIDTH (1 IS WIDEST),
   AND 3RD IS COLOR REGISTER, 0-4).
```

PROGRAM.095:Color Static

Here's an interesting variation of PROGRAM.094. Try it, you'll like it!

P1 Time duration of effect.

P2 Code number for color register.

```
0 REM PROGRAM.095
10 GRAPHICS 2+16:POSITION 6,2:? #6;"Ma
██I█e":POSITION 5,7:? #6;"l██G█u█E!"
20 FOR N=0 TO 4
30 X=USR(ADR("hhh██hh██<█▼█●█▶██▶██ █ ██
●██████♦"),3,N)
40 NEXT N
```

PROGRAM.096 Two Number Comparisons

If the first number is less than the second number, the Return Variable will contain a 1. If they are equal, a 2 will be returned. If the first number is greater than the second one, a 3 will be returned. This type of comparison makes a nice setup for a ON...GOTO or ON...GOSUB statement, by eliminating three IF...THEN statements. The two parameters required are the two integer numbers being compared.

```
0 REM PROGRAM.096
10 A=20
20 B=99
30 X=USR(ADR("h[illegible]hh[illegible]hh[illegible]
[illegible]"),A,B):REM MAXIMUM PARAMETER
   SIZE IS 255.
40 ? X
50 END
60 REM *
70 REM USE FOLLOWING ROUTINE FOR
   NUMBERS UP TO 65535.
80 X=USR(ADR("h[illegible]h[illegible]h[illegible]h[illegible]h[illegible]
[illegible]"),A,B)
```

PROGRAM.097 Cursor Blink

This program will blink the cursor or any inverse character. The blinking is set up in a Vertical Blank Interrupt. The routine can not be addressed directly but must be placed in a knowm area of memory. Use PROGRAM.092 to clear this routine.

P1 Starting Address of the VBI machine code.

```
0 REM PROGRAM.097
10 DIM VBI$(38)
20 VBI$="hh[illegible]h[illegible]i [illegible] \[illegible]n[illegible] In[illegible] Ijjjj
.[illegible] I.[illegible] ILb[illegible]"
30 X=USR(ADR(VBI$),ADR(VBI$))
```

PROGRAM.098 Low/High Byte Of A Number

This routine will return the Low byte of an integer number in the Return Variable and the High byte in memory location 208. The maximum value of the number is 65535.

P1 Integer number to determine low & high byte values.

```
0 REM PROGRAM.098
10 A=2555
20 X=USR(ADR("hh[graphics]h[graphics]♥[graphics]◆"),A)
30 ? X,PEEK(208)
```

PROGRAM.099 Peek A Two Byte Register

This program will read two consecutive memory locations in the standard low/high byte convention and return the full address in the Return Variable. It can be useful for repeated readings of low and high byte registers, such as the Display List and Screen Memory starting addresses.

The program below returns the starting address of the screen memory. Line 20 prints out the full address, which is the memory location of the upper left corner of the screen. POKEing a '64' into this memory location places the heart character in that corner (remember to use the Display value - see Appendix). Only the first memory address (low byte) of the two byte address is required, as the single parameter.

P1 Low byte of a two byte address.

```
0 REM PROGRAM.099
10 X=USR(ADR("hhh[graphics]♥[graphics]◆"),88
)
20 ? X:POKE X,64
```

PROGRAM.100 Poke A Two Byte Register

This program does just the opposite of PROGRAM.099. It will place an integer number up to 65535 into a two byte, low byte then high byte, address.

The sample program shows the placing of the number 40201 into Page 6 locations 1536 and 1537. Line 20 prints out the full address in BASIC of the 2 byte address to verify the number was correctly inserted.

P1 First location of a 2 byte address.
P2 Number to place into the two addresses.

```
0 REM PROGRAM.100
10 X=USR(ADR("hh[illegible]"),1536
,40201)
20 ? PEEK(1536),PEEK(1537)
30 ? PEEK(1536)+PEEK(1537)*256
```

PROGRAM.101 Time It

These two routines will time an event in your BASIC program. The first routine in line 10 turns on the timer. The second routine in line 30 turns off the timer. The elapsed time is in the Return Variable. This time is in Jiffies which is approximently 1/60th of a second. The routine will be good for up to 65535 jiffies or about 1092 seconds. No parameters are required. The sample program shows the timing of a delay loop in line 20.

```
0 REM PROGRAM.101
10 START=USR(ADR("h[illegible]"))
20 FOR I=1 TO 300:NEXT I
30 TIME=USR(ADR("h[illegible]"))
40 ? TIME
50 REM TIME IS IN JIFFIES OR APROX.
   1/60 OF A SECOND.  FOR SECONDS
   JUST DIVIDE BY 60.
```

PROGRAM.102 Delay Timer

This short routine will cause a delay in processing for up to about 4.5 seconds. The single parameter is the time delay in jiffies, 0-255.

```
0 REM PROGRAM.102
10 X=USR(ADR("hhh[illegible]"),2
55)
20 REM PARAMETER IS NUMBER UP TO 255
   WHICH IS APPROXIMENTLY 4.5 SECONDS
   OF DELAY TIME.
```

PROGRAM.103 Disable/Able Break Key

To disable the Break Key, use the routine in line 10. To re-enable the Break Key use the routine in line 30. Remember the Break Key is also enabled whenever a new GRAPHICS command is called, so you may want to use the disable routine several times thoughout your program.

```
0 REM PROGRAM.103
10 X=USR(ADR("h[illegible]"))
20 FOR I=1 TO 1000:NEXT I
30 X=USR(ADR("h[illegible]"))
```

PROGRAM.104 Sound Off

You can use this short machine routine to turn off all four SOUND registers at once. Lines 10 to 40 turn on the sound registers, and line 60 turns them off.

```
0 REM PROGRAM.104
10 SOUND 0,10,10,6
20 SOUND 1,20,8,6
30 SOUND 2,100,6,6
40 SOUND 3,160,14,6
50 FOR I=1 TO 500:NEXT I
60 X=USR(ADR("h[illegible]"))
```

PROGRAM.105 Code/Decode A Line Of Text.

There may be times when you don't want your text lines readable in a BASIC program. This occurs many times in adventure game programming. You can code the text, and have the text decoded in the program using BASIC, but this process takes too much time. The following program will code or decode a text string up to 256 characters long in the flash of an eye!

The first routine in line 30 codes the text in line 20. It codes it to an unreadible form, as line 50 shows. The same routine is used in line 60 to decode the coded string. Three parameters are used.

- P1 Starting address of the text.
- P2 Length of the text.
- P3 Seed number (1-255) for coding or decoding. Must be the same for both routines.

```
0 REM PROGRAM.105
10 DIM S$(256)
20 S$="THE CAT IN THE HAT"
30 X=USR(ADR("hh[illegible]h[illegible]hh[illegible]hh[illegible]E
[illegible]"),ADR(S$),LEN(S$),4)
40 POKE 766,1:REM PRINT CONTROL
   CHARACTERS WITHOUT ACTING ON THEM.
50 ? S$:REM CODED.
60 X=USR(ADR("hh[illegible]h[illegible]hh[illegible]hh[illegible]E
[illegible]"),ADR(S$),LEN(S$),4)
70 ? S$:REM DECODED.
80 END
90 REM *
100 REM USE THE FOLLOWING ROUTINE FOR
    UP TO 65535 LENGTH.
110 X=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]hh[illegible]
[illegible]E[illegible]E[illegible]"),P1,P2,
P3)
```

Chapter 14
Number Systems Conversion

PROGRAM.106 Decimal To Binary

This program will convert a decimal integer number, up to 255, into its binary equivalent. Line 20 sets up a string of eight spaces for the binary number.

P1 Decimal Integer Number, <256.
P2 Address of string for Binary number.

```
0 REM PROGRAM.106
10 DIM BIN$(8)
20 BIN$="        ":REM MUST BE EIGHT
   SPACES.
30 X=USR(ADR("hhh[illegible]h[illegible]h[illegible]\F[illegible]H[illegible]0[illegible] [illegible]1[illegible]
[illegible]"),243,ADR(BIN$))
40 ? BIN$
```

PROGRAM.107 Decimal To Hexadecimal

To convert a decimal integer to its hexadecimal equivalent, use the next program. Note, the string to hold the hexadecimal number must be set up as shown in line 20 for decimal numbers up to 255, and as shown in line 70 for decimal numbers up to 65535.

P1 Decimal Integer Number
P2 Address of string for hexadecimal number.

```
0 REM PROGRAM.107
10 DIM HEX$(5)
20 HEX$="$   ":REM MUST DO THIS.
30 DEC=243
40 X=USR(ADR("hhh[illegible]h[illegible]h[illegible]) [illegible]
i 7[illegible]i 0[illegible]JJJJ[illegible]"),DEC,ADR(HE
X$))
50 ? HEX$
60 END
70 REM USE FOLLOWING ROUTINE FOR
   DECIMAL NUMBERS UP TO 65535.  BE
   SURE TO SET HEX$="$     ".
80 X=USR(ADR("hh[illegible]h[illegible]h[illegible]h[illegible]) [illegible]
[illegible]i 7[illegible]i 0[illegible]JJJJ[illegible]
[illegible]"),P1,P2)
```

PROGRAM.108 Hexadecimal To Decimal

This program does the opposite of the last program. The hex number is placed in a string, without the '$' sign, and the Return Variable contains the decimal equivalent.

P1 Address of the hexadecimal number.

```
0 REM PROGRAM.108
10 DIM HEX$(4)
20 HEX$="FE"
30 DEC=USR(ADR("hh[illegible]h[illegible]A[illegible]
[illegible]7[illegible]8[illegible]0[illegible]"),ADR(HEX
$)):REM MAXIMUM HEX NUMBER = $FF.
40 ? DEC
50 END
60 REM USE FOLLOWING ROUTINE FOR HEX
   NUMBERS UP TO $FFFF.
70 X=USR(ADR("hh[illegible]h[illegible]A[illegible]7[illegible]
[illegible]8[illegible]0[illegible]")
,P1)
```

PROGRAM.109 Hexadecimal To Binary

This number conversion requires two strings to hold the numbers. The binary string must be reserved with eight spaces as shown in line 20. The other string contains the hex number to convert.

P1 Address of the hexadecimal string.
P2 Address of the binary string.

```
0 REM PROGRAM.109
10 DIM HEX$(2),BIN$(8)
20 BIN$="        ":REM MUST BE 8
   SPACES.
30 HEX$="FE"
40 X=USR(ADR("hh[illegible]
7[illegible] 8[illegible]0[illegible]
0[illegible] [illegible]1[illegible]"),ADR(HEX$),ADR(BIN$))
50 ? BIN$
```

PROGRAM.110 Binary To Decimal

To convert an eight bit binary number to its decimal equivalent, use this program. The binary number is placed in a string, and the Return Variable contains the decimal number.

P1 Address of the binary number.

```
0 REM PROGRAM.110
10 DIM BIN$(8)
20 BIN$="11110110"
30 DEC=USR(ADR("h[illegible]h[illegible]h[illegible]
0[illegible]e[illegible]"),ADR(BIN$))
40 ? DEC
```

PROGRAM.111 Binary To Hexadecimal

Two strings are required to do this number conversion. The binary number maximum is 11111111 ($FF). The string to hold the hex number must be set up as shown in line 20. As an extra bonus, the decimal equivalent of the binary number is also available in the Return Variable.

P1 Address of the binary number string.
P2 Address of the hexadecimal number string.

```
0 REM PROGRAM.111
10 DIM B$(8),H$(3)
20 H$="$  ":REM MUST DO THIS.
30 B$="11111110"
40 X=USR(ADR("h[illegible]h[illegible]h[illegible]h[illegible]h[illegible]
[illegible]0[illegible]e[illegible]i 7[illegible]i
0[illegible]JJJJ[illegible]"),ADR(B$),ADR(H$))
50 ? H$,X
```

Chapter 15
Bit Flipping, Reading, Clearing, & Setting

The next set of routines will come in handy for more advanced BASIC programmers and machine language programmers. A bit in a binary number is set when its value is '1' and cleared, or not set, when it is '0'. The maximum 8-bit binary number is 11111111, which is 255 decimal, or $FF hex.

PROGRAM.112 Bit Flip - Number In A Memory Location.

This program allows any of the eight bits in a binary number to be changed to its opposite value. The number is in a known memory location. The sample program places the decimal number 12 in memory location 1536, Page 6, where the machine routine will act upon it.

P1 Address of number.
P2 Bit to flip, 0-7.

```
0 REM PROGRAM.112
10 POKE 1536,12
20 X=USR(ADR("hh[illegible]h[illegible]hh[illegible]8X[illegible]Q[illegible]"
),1536,4)
30 ? PEEK(1536)
40 REM EXAMPLE SHOWS THE NUMBER "12"
   BECOMES '28' WHEN BIT 4 IS FLIPPED.
```

PROGRAM.113 Bit Flip - Direct Number Input.

This routine does the same bit flipping as the last program, but the number is read directly into the machine routine as a parameter. The new number is found in the Return Variable.

P1 Integer Number, 0-255.

P2 Bit to flip, 0-7.

```
0 REM PROGRAM.113
10 A=28
20 X=USR(ADR("hhh[illegible]hh[illegible]8X[illegible]E[illegible]")
,A,4)
30 ? X
```

PROGRAM.114 Bit Set - Number In Memory Location

To set a bit of a number in a memory location, use this next routine.

P1 Address of number.

P2 Bit to set, 0-7.

```
0 REM PROGRAM.114
10 POKE 1536,2
20 X=USR(ADR("hh[illegible]h[illegible]hh[illegible]8X[illegible]"
),1536,3)
30 ? PEEK(1536)
40 REM EXAMPLE SHOWS THE NUMBER '2'
   BECOMES '10' WHEN BIT 3 IS SET.
```

PROGRAM.115 Bit Set - Direct Number Input

To set any bit of a number fed directly into the machine routine use the next program. The new number will be in the Return Variable.

P1 Integer Number, 0-255.

P2 Bit to Set, 0-7.

```
0 REM PROGRAM.115
10 A=2
20 X=USR(ADR("hhh[illegible]hh[illegible]♥[illegible]8×[illegible]h [illegible]◆")
,A,3)
30 ? X
```

PROGRAM.116 Bit Clear - Number In Memory Location

To clear a bit in a number stored in memory use this routine.

P1 Address of number.
P2 Bit to clear, 0-7.

```
0 REM PROGRAM.116
10 POKE 1536,255
20 X=USR(ADR("hh[illegible]h[illegible]hh[illegible]♥[illegible]×[illegible]1[illegible]◆
"),1536,1)
30 ? PEEK(1536)
40 REM :EXAMPLE SHOWS THE NUMBER '255'
   BECOMES '253' WHEN BIT 1 IS CLEARED
```

PROGRAM.117 Bit Clear - Direct Number Input

To clear a bit in a number read directly into the machine program use this next routine. The new number is assigned to the Return Variable.

P1 Integer Number, 0-255.
P2 Bit to clear, 0-7.

```
0 REM PROGRAM.117
10 A=255
20 X=USR(ADR("hhh[illegible]hh[illegible]♥[illegible]×[illegible]◆
"),A,1)
30 ? X
```

PROGRAM.118 Bit Read - Number In Memory Location

To read the status of a bit in a number in a memory location, use the next program. The status, 0 or 1, of the desired bit is assigned to the Return Variable.

P1 Address of number.
P2 Bit to Read, 0-7.

```
0 REM PROGRAM.118
10 POKE 1536,28
20 X=USR(ADR("hh[illegible]h[illegible]hh[illegible]♥[illegible]0[illegible]j[illegible])
[illegible]◆"),1536,2)
30 ? X
40 REM :EXAMPLE SHOWS THE NUMBER '28'
   HAS BIT 2 SET(EQUAL TO '1').
```

PROGRAM.119 Bit Read - Direct Number Input

This is the companion to the previous program when you want to input the number directly to the machine program. The Return Variable contains the status of the desired bit, 0 or 1.

P1 Integer Number, 0-255.
P2 Bit to read, 0-7.

```
0 REM PROGRAM.119
10 A=28
20 X=USR(ADR("hhh[illegible]hh[illegible]♥[illegible]0[illegible]j[illegible]) [illegible]
[illegible]◆"),A,2)
30 ? X
```

Chapter 16
Text on a Graphics 8 Screen

PROGRAM.120 Text On Graphics 8 Screen

You can place normal GRAPHICS 0 text on a GRAPHICS 8 picture screen with the following routine. The program works by first reading in the ATASCII value for the text you want to place on the screen. Then the program goes to that character in the internal Atari character set and copies its shape directly to the screen memory.

Line 20 sets up a GRAPHICS 8 screen with a black background and red border. Line 50 selects the ATASCII value of 65 which is the letter 'A' character. Line 60 through 70 runs through the machine routine 5 times in order to place the letter 'A' in each of the four corners and at the start of the second line.

The first parameter is the ATASCII number of the character to place on the screen and the second parameter is the location offset on the screen to place the character. Line 100 contains the second parameter for the example program. This offset can start at 0 (upper left corner of screen) and continue to 7399 (lower right corner of screen). But isn't a GRAPHICS 8 screen 7680 bytes in length? Yes it is but since a text character is 8 lines high and 8 bits wide then we have to back off from the very last byte of the screen to fit it in the corner.

You can put 40 characters across the screen and start them on any of the 192 lines of the screen. Therefore, for correct spacing, the first character on the second line would start on line 8 (remember the first line is line 0) and have a location offset of 8x40 or 320.

P1 ATASCII value of character to place on screen.
P2 Location offset on screen to place character.

```
0 REM PROGRAM.120
10 DIM P$(125)
20 GRAPHICS 8+16:POKE 710,0:POKE 712,6
6
30 P$(1)="h[illegible]heY[illegible]heX[illegible]i [illegible]hh[illegible]
[illegible]8[illegible] [illegible]i @[illegible] [illegible]8[illegible]
[illegible]1[illegible]i ([illegible]i [illegible]"
40 P$(102)="[illegible]I[illegible]"
50 NUMBER=65:REM ATASCII CODE FOR 'A'.
60 FOR I=1 TO 5:READ LOCATION
70 BB=USR(ADR(P$),LOCATION,NUMBER)
80 NEXT I
90 GOTO 90
100 DATA 0,39,7360,7399,320
```

Appendix A
ATASCII and Display Character Code Values

Character	ATASCII	Display
space	32	0
!	33	1
"	34	2
#	35	3
$	36	4
%	37	5
&	38	6
'	39	7
(	40	8
)	41	9
*	42	10
+	43	11
,	44	12
-	45	13
.	46	14
/	47	15
0	48	16
1	49	17
2	50	18
3	51	19
4	52	20
5	53	21
6	54	22
7	55	23
8	56	24
9	57	25
:	58	26
;	59	27
<	60	28
=	61	29
>	62	30
?	63	31
@	64	32
A	65	33
B	66	34
C	67	35
D	68	36
E	69	37
F	70	38
G	71	39
H	72	40
I	73	41
J	74	42
K	75	43
L	76	44
M	77	45
N	78	46
O	79	47
P	80	48
Q	81	49
R	82	50
S	83	51
T	84	52
U	85	53
V	86	54
W	87	55
X	88	56
Y	89	57
Z	90	58
[	91	59
\	92	60
]	93	61
^	94	62
_	95	63
CTRL ,	0	64
CTRL A	1	65

Character	ATASCII	Display	Character	ATASCII	Display
CTRL B	2	66	a	97	97
CTRL C	3	67	b	98	98
CTRL D	4	68	c	99	99
CTRL E	5	69	d	100	100
CTRL F	6	70	e	101	101
CTRL G	7	71	f	102	102
CTRL H	8	72	g	103	103
CTRL I	9	73	h	104	104
CTRL J	10	74	i	105	105
CTRL K	11	75	j	106	106
CTRL L	12	76	k	107	107
CTRL M	13	77	l	108	108
CTRL N	14	78	m	109	109
CTRL O	15	79	n	110	110
CTRL P	16	80	o	111	111
CTRL Q	17	81	p	112	112
CTRL R	18	82	q	113	113
CTRL S	19	83	r	114	114
CTRL T	20	84	s	115	115
CTRL U	21	85	t	116	116
CTRL V	22	86	u	117	117
CTRL W	23	87	v	118	118
CTRL X	24	88	w	119	119
CTRL Y	25	89	x	120	120
CTRL Z	26	90	y	121	121
ESCAPE	27	91	z	122	122
CTRL ↑	28	92	CTRL ;	123	123
CTRL ↓	29	93			124	124
CTRL ←	30	94	CLEAR	125	125
CTRL →	31	95	Back S	126	126
CTRL .	96	96	TAB	127	127

The following special characters, along with the Arrow, Clear, Insert, Back S, Tab, and Escape keys must be preceeded by the ESC key to display

CTRL 2	253	253	CTRL Insert	255	255
CTRL Back S	254	254	E.O.L.	155	155

For inverse characters, add 128 to the above values.

Appendix B
Atari Text & Graphic Modes

	Antic Mode	BASIC Mode	Columns	Rows	Number of Colors	Bytes per Line	Screen Memory Bytes
TEXT							
	2	0	40	24	1[2]	40	960
	3		40	**	1[2]	40	**
	4	12*	40	24	5	40	960
	5	13*	40	12	5	40	480
	6	1	20	24	5	20	480
	7	2	20	12	5	20	240
GRAPHICS							
	8	3	40	24	4	10	240
	9	4	80	48	2	10	480
	A	5	80	48	4	20	960
	B	6	160	96	2	20	1920
	C	14*	160	192	2	20	3840
	D	7	160	96	4	40	3840
	E	15*	160	192	4	40	7680
	F	8	320	192	1[2]	40	7680
GTIA							
	9	9	80	192	1[3]	40	7680
	10	10	80	192	10	40	7680
	11	11	80	192	16[4]	40	7680

[1] Full Screen, No Window

[3] One Color, 16 Luminances

* XL/XE Only

[2] One Color, 2 Luminances

[4] 16 Colors, One Luminance

** User Defined

BASIC TURBOCHARGER *Source Codes*

2-Disk set with the complete source codes for all the routines and programs from *BASIC TURBOCHARGER* $15.00

Other 8-Bit Products From Alpha Systems

Atari Software Protectecion Techniques and Advanced Atari Protection Techniques
Learn the secrets of software protection including routines to protect your own programs, and reviews of commercial back-up software - Both books and disks $39.95

Chipmunk Disk Back-Up System - Creates working back-up copies of most software - No hardware modifications needed! (write for full list of what it will back-up) $34.95

Scanalyzer - Scan, analyze & modify any program $29.95

Impersonator - Cartridge back- up system $29.95

Magniprint II+ - Prints almost any picture from 1/8 pg to 6'. Add multi-size text, any font, choose your own print shades $24.95

Magniprint II+ & ComputerEyes - Complete video digitizer and print package (Gr. 9 software $12.00 extra) $119.95

Graphics Transformer - Change pcitures from one format to another. Use ComputerEyes pictures in Print Shop! $22.95

Parrot - Sound digitizer you have to hear to believe! $59.95

Your Atari Comes Alive - Create exciting computer controlled devices, incl. speech recognition & more! book & disk $24.95

Call or write for free catalog of all our fine 8-bit products
Add $3.00 shipping/handling per order. Ohio res. add 5 1/2% tax.

ALPHA SYSTEMS 1012 Skyland Dr.
Macedonia, OH 44056
Order Line (216) 374-7469

LIMITED WARRANTY

Alpha Systems warrants the original purchaser of this computer software product that the recording medium on which the software programs are recorded will be free from defects in materials and workmanship for ninety days from the date of purchase. Defective media returned by the purchaser during that ninety day period will be replaced without charge, provided that the returned media have not been subjected to misuse, damage, or excessive wear.

Following the initial ninety day warranty period, defective media will be replaced for a replacement fee of $6.50.

Defective media should be returned to:

ALPHA SYSTEMS
1012 SKYLAND DRIVE
MACEDONIA, OH. 44056

in protective packaging accompanied by: (1) a brief statement describing the defect; (2) a $6.50 check or money order (if beyond the ninety day warranty period); (3) your return address; (4) the problem disk.

What is Not Covered by this Warranty

This warranty does not apply to the software programs themselves. the programs are provided "as is".

This warranty is in lieu of all other warranties, whether oral or written, express or implied. Any implied warranties, including imputed warranties of merchantability and fitness for a particular purpose, are limited in duration to ninety days from the date of purchase. Alpha Systems shall not be liable for incidental or consequential damage for breach of any express or implied warranty.

The provisions of the foregoing warranty are subject to the laws of the state in which the disk is purchased. Such laws may broaden the warranty protection available to the purchaser of the disk.

Tell Us What You Think

We at Alpha Systems are sincerely interested in bringing you the best possible products at the lowest possible prices. Please write us if you experience any difficulties with our products, or have any comments or ideas for improvements. We will do our best to make our products better meet your needs. When you write, please enclose the following: 1) Your name, address, and phone number. 2) Your comments, or a description of your problem. 3) A description of your system. 4) If you are reporting a problem, please also include a description of what you were doing when the problem occurred, any printouts or other output showing the problem if possible, and any suggestions you may have regarding the cause and solution.

BASIC

TURBOCHARGER

By Jeff Bader

Machine Language Routines
For Atari BASIC Programmers

TURBOCHARGE your BASIC programs! Now anyone, even beginning BASIC programmers, can put the **speed, power,** and **flexibility** of machine language into any BASIC program. This unique book and disk package contains over 160 **ready to run** machine language routines that can easily be used in any BASIC program. These routines have been written, tested, and used by some of the best professional programmers. Just a few of the many topics covered are:

Moving Memory	Player/Missiles
Graphics Modes	Sorting & Searching
String Search	Disk Input/Output
Timing Routines	Joystick Routines
DLI Routines	Bit Manipulations
Graphics & Text	Special Effects
Many More!	

Each routine is listed and completely explained in the book, and included on the disk. They're ready to use, **no typing required!** All the work has been done for you. There's no license fees, so you can use these routines in your own commercial programs. Don't wait - **TURBOCHARGE** your BASIC programs today!

ALPHA SYSTEMS