

COMPUTE!'s THIRD BOOK OF ATARI

Games, utilities, tutorials and other helpful
information for users of Atari® personal computers.

XL Compatible

A **COMPUTE! Books** Publication

\$12.95

COMPUTE!'s

THIRD BOOK

OF

ATARI

COMPUTE!™ Publications, Inc. 

One of the ABC Publishing Companies

Greensboro, North Carolina

Atari is a registered trademark of Atari, Inc.

The following article was originally published in *COMPUTE!* Magazine, copyright 1981, Small System Services, Inc.:

"Blinking Characters" (December)

The following articles were originally published in *COMPUTE!* Magazine, copyright 1982, Small System Services, Inc.:

"Machine Language Sort" (March)

"Elementary Numbers" (October)

"The Atari Wedge" (November)

"Purge" (November)

"Atari PEEK and POKE Alternative" (December)

"CalCalc" (December)

The following articles were originally published in *COMPUTE!* Magazine, copyright 1983, Small System Services, Inc.:

"Atari Exponents" (January)

"Automate Your Atari" (January)

"The Atari Cruncher" (February)

"SuperFont Plus" (February)

"16-Bit Atari Music" (March)

"Scriptor" (April)

"Atari Starshots" (May)

The following articles were originally published in *COMPUTE!* Magazine, copyright 1983, *COMPUTE!* Publications, Inc.:

"Using the Atari Timer" (June)

"Laser Gunner II" (July)

"Circles" (July)

"Castle Quest" (July)

"Atari Sound Experimenter" (July)

"Atari Verify" (August)

"Spelling Quiz" (October)

"String Arrays In Atari BASIC" (November)

Copyright 1984, *COMPUTE!* Publications, Inc. All rights reserved.

Reproduction or translation of any part of this work beyond that permitted by Sections 107 and 108 of the United States Copyright Act without the permission of the copyright owner is unlawful.

Printed in the United States of America

ISBN 0-942386-18-3

10 9 8 7 6 5 4 3 2

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is a subsidiary of American Broadcasting Companies, Inc., and is not associated with any manufacturer of personal computers. Atari is a trademark of Atari, Inc.

Contents

Foreword	v
Chapter 1: Programming Hints	1
Exponents	
<i>Matt Giwer</i>	3
Reading the Keyboard Codes	
<i>Orson Scott Card</i>	4
Using the Atari Timer	
<i>Stephen Levy</i>	22
Blinking Characters	
<i>Frank C. Jones</i>	27
String Arrays	
<i>Stephen Levy</i>	31
Chapter 2: Sound	35
Sound Experimenter	
<i>Matt Giwer</i>	37
16-Bit Music	
<i>Fred Tedsen</i>	45
Chapter 3: Applications and Education	53
Beginner's Keyboard	
<i>Marty Albers</i>	55
Spelling Quiz	
<i>Edward Perrin</i>	57
Elementary Numbers	
<i>Stephen Levy</i>	66
Standings	
<i>Dan and Philip Seyer</i>	74
CalCalc: Computerize Your Diet	
<i>Charles Brannon</i>	87
Castle Quest	
<i>Timothy G. Baldwin</i>	94
Scriptor: An Atari Word Processor	
<i>Charles Brannon</i>	102
Chapter 4: Graphics	125
SuperFont Plus	
<i>John Slaby and Charles Brannon</i>	127
Super TextPlot	
<i>Donald L. Vossler</i>	142
Circles	
<i>Jeffrey S. McArthur</i>	153

Chapter 5: Utilities	161
Joystick Cursor Control	
<i>Jeff Brenner</i>	163
Atari Verify	
<i>Michael J. Barkan</i>	165
Automate Your Atari	
<i>Joseph J. Wrobel</i>	167
The Wedge: Adding Commands To Atari BASIC	
<i>Charles Brannon</i>	174
Renumber Plus	
<i>Manny Juan and Paul N. Havey</i>	191
Purge	
<i>Al Casper</i>	195
Chapter 6: Advanced Techniques	199
Starshot	
<i>Matt Giwer</i>	201
Laser Gunner II	
<i>Gary R. Lecompte</i>	216
The Cruncher	
<i>Andrew Lieberman</i>	225
PEEK and POKE Alternatives	
<i>Jerry White</i>	228
Chapter 7: Beyond BASIC	231
1200 Memory Map: An Initial Examination	
<i>Ian Chadwick</i>	233
Merging Machine Language into BASIC	
<i>Fred Pinho</i>	248
Machine Language Sort Utility	
<i>Ronald and Lynn Marcuse</i>	258
Appendix A: A Complete Guide to the Atari	
Character Set	275
Appendix B: A Beginner's Guide to Typing	
In Programs	301
Appendix C: How to Type In Programs	305
Index	306

Foreword

Like COMPUTE!'s other books devoted to the Atari home computer, *COMPUTE!'s Third Book of Atari* is packed with articles on programming techniques, ready-to-run software, computer utilities, and reference information—all designed to make your Atari computer even more useful than before.

Whether you are a beginner or an advanced programmer, you will find numerous articles of interest, ready to type into your computer, games and applications designed to help you get more from your investment, and helpful hints and utilities to help you better understand your Atari.

If you already have COMPUTE!'s First and Second Books of Atari, you know just how valuable they are—how often you open the books to look up the information you need to meet your own programming challenges. However, if this is your first COMPUTE! book, you're in for some pleasant surprises.

SECRET

CONFIDENTIAL - This document contains information of a confidential nature and its disclosure to unauthorized persons could result in the identification of sources and methods of the Central Intelligence Agency and the Department of Defense. It is to be controlled, stored, handled, transmitted, and disposed of in accordance with the policies and procedures of the Central Intelligence Agency and the Department of Defense. It is to be destroyed when it is no longer needed for official use.

1 Programming Hints

1

Exponents

Matt Giwer

The exponential operator, ^, can be made accurate and useful. Here's how.

The exponential operator, ^, performs a very standard mathematical function, although if you are not familiar with mathematics you may not be aware of its potential. Also, there is another byte-saving use that I will save for the end.

The key to making full use of ^ is to realize that in mathematical notation the square root of four is the same as four to the one-half power. In BASIC you can write either `SQR(4)` or `4^(1/2)`. So what good is that? Well, you might want to do a cube root, which would be `8^(1/3)`. Get the idea? Not believing that this works, you might have tried it by now and have noticed that the machine insists that `4^(1/2)` is not 2 but rather 1.998... something. It seems strange to accept a wrong answer from a very slow function.

To correct for this inaccuracy, we simply write the instruction `INT (4^(1/2) + 0.01)`, and this will return the number 2. In return for this inaccuracy we get the ability to calculate very unusual powers and roots. The above could have been written `4^0.5` and the same answer returned. We could just as easily have written `4^0.4321` or `2^2.223` and have gotten an answer correct enough for many calculations. Also, those complex problems such as two to the five-thirds power `2^(5/3)` can be calculated with ease. So not only can we do the more common cube roots by using `^(1/3)`, but we can now also do an entire range of mathematical functions.

It is not only faster but more accurate to write `2*2` rather than `2^2`. If we are not doing mathematics, how do we make use of this? How about instead of writing a byte-consuming timing loop for a beep, we simply write `A = 1^1`? If the beep should last longer, then there is always `A = 1^1^1^1^1^1`, etc. It takes quite a while before this simple statement equals the number of bytes consumed by a timing loop. Thus the major drawback to more frequent use of ^ can be turned to our advantage.

Reading the Keyboard Codes

Orson Scott Card

By reading the Atari keyboard directly, you can get almost any key to perform like a function key—without changing any of the regular uses of the keyboard.

Whenever you press a key on your Atari keyboard, a number is stored at location 53769 in memory and, in most cases, in a shadow register at location 764. That number is the keyboard code for the key, or combination of keys, you pressed.

Unfortunately, that number has no relation at all either to ATASCII character code or to the Atari's internal character code. So most programmers ignore the keyboard code (KEYCODE) and let the operating system translate the keyboard code into ATASCII form.

You can use the KEYCODE, however, to get some interesting results:

Speed. Picking up the keyboard code at 53769 or 764 can save you time, especially when you're working in machine language. For one thing, you completely short-circuit the "debounce" routine that makes the computer wait for a while before repeating a key that is being held down continuously. If the key is down, it's down, and you can read the value at once. That can be a disadvantage if you have a touch-typing program, but it can be a great help if you *want* instant repetition of a key.

Customization. You can set up your computer, with software, to read the keys any way you like. This article, for instance, includes a program to make your computer read the keyboard according to the Dvorak pattern instead of the standard Qwerty layout. Also, you can set up your own system for shift-locking the keyboard. You don't have to follow the standard computer system of locking and unlocking only the alphabetic characters when you press SHIFT-CAPS/LOWR, CONTROL-CAPS/LOWR, or CAPS/LOWR alone. You can make the entire keyboard lock and unlock, or have the nonalphabetic characters lock independently of the

alphabetic characters, by pressing SHIFT-ESCAPE or CONTROL-ESCAPE, for instance.

Range. Perhaps the most exciting advantage of working with the keyboard code is the great range of values it offers you. Every key on the keyboard except SHIFT, CONTROL, BREAK, START, SELECT, OPTION, and RESET produces its own unique KEYCODE number. Holding down SHIFT while depressing another key produces that *same* number plus 64. Holding down CONTROL produces that number plus 128. And, except for 11 keys (16 for XL users), holding down *both* SHIFT and CONTROL produces that number plus 192.

This means almost every key has four possible values—even RETURN and ESC and SPACE, which the computer usually treats the same regardless of whether SHIFT or CONTROL is pressed. There are 52 keys on old Ataris and 57 keys on XL models that put numbers in location 53769. That gives you 197 unique signals from your keyboard (212 if you use an XL model).

Yet there are only 128 valid ATASCII codes (values 128-255 are merely inverse characters). You are left with 68 (or 84) possible key combinations that ATASCII doesn't need to use. If you were creating a word-processing program, you could print every single character, including graphics characters, and still have 68 commands left over—without ever reaching for the console keys.

The Three Atari Character Codes

The Atari Operating System (OS) uses three different codes for character values: ATASCII, Internal Code (ICODE), and Keyboard Code (KEYCODE). Each has a specific use, and most of the time, the OS handles all the conversions from one to another so quickly that you don't even notice it's going on.

In order to use KEYCODEs effectively, you need to have a clear idea of the differences among the three codes and their relationship to each other. So let's review the function of each of the character codes.

ATASCII. This is the code used by BASIC. All the alphanumeric characters (letters and numbers) and symbols follow the standard ASCII code recognized by most computers. The rest of the ATASCII codes are used for graphics characters. For instance, in ATASCII, the letter A has the value 65.

The following commands and functions use the ATASCII number:

```
CHR$(ATASCII)
OPEN #1,4,0,"K:":GET #1,ATASCII
ATASCII=PEEK(763)
ATASCII=ASC("A")
GRAPHICS 1:COLOR ATASCII:PLOT I,I
```

(Special ATASCII code conversions are used in GRAPHICS 1 and 2, but for values 32-95, the regular ATASCII values will PLOT in Color 1—color register 0.)

Internal code. This is the code used by the operating system to put characters on the screen. The ICODE (internal code) number represents the character's position within the ROM character set. The first character in the ROM character set is the blank (space) character. It has the ICODE number 0. The character A is in position 33 in the character set, so its ICODE number is 33.

The ICODE number is used twice. First, when you type or PRINT a character on the screen, the OS converts the ATASCII value into the ICODE value and stores the ICODE value in screen memory. Second, the ANTIC chip, which scans screen memory 60 times a second, reads the ICODE value stored there and uses it to count a certain number of steps into the ROM character set. Since it takes eight bytes to contain each character pattern in the ROM set, ANTIC counts $8 \times \text{ICODE}$ bytes into the character set to find the beginning of the pattern.

So when you type the letter A, the OS stores the number 33 in screen memory. ANTIC finds that 33 and multiplies it by 8, which results in the number 264. ANTIC then goes to the character set and counts in until it finds byte 264. This is the first byte of the pattern for the character A. ANTIC uses that byte, along with the next seven bytes, to tell the TV screen what to display.

You will use ICODE values for the same purpose the OS uses them—to POKE characters directly into screen memory and to find a character's pattern within the character set.

Keyboard code. This is the number generated by the circuits in your keyboard when you press a key (see Table 1). The combination of open and closed circuits from the keyboard causes a KEYCODE (keyboard code) number to be stored in location 53769. This number is then read by the OS and stored at 764, where it is picked up and converted into an ATASCII value which is stored in location 763.

The keyboard code is never used anywhere else, but there are still several things you can do with it. By POKEing character codes into location 764, you can fool the OS into thinking that a particular key has been pressed. Then, when your program GETs

Table 1. Keyboard Codes**Unshifted Keyboard Values**

ESC	1	2	3	4	5	6	7	8	9	0	<	>	DEL
28	31	30	26	24	29	27	51	53	48	50	54	55	52
TAB	Q	W	E	R	T	Y	U	I	O	P	-	=	RETURN
44	47	46	42	40	45	43	11	13	8	10	14	15	12
	A	S	D	F	G	H	J	K	L	;	+	*	CAPS/LOWR
	63	62	58	56	61	57	1	5	0	2	6	7	60
	Z	X	C	V	B	N	M	,	.	/			Atari logo
	23	22	18	16	21	35	37	32	34	38	39		
SPACE BAR		F1	F2	F3	F4	HELP							
33		3	4	19	20	17							

Keyboard Values with SHIFT*

ESC	1	2	3	4	5	6	7	8	9	0	<	>	DEL
92	95	94	90	88	93	91	115	117	112	114	118	119	116
TAB	Q	W	E	R	T	Y	U	I	O	P	-	=	RETURN
108	111	110	106	104	109	107	75	77	72	74	78	79	76
	A	S	D	F	G	H	J	K	L	;	+	*	CAPS/LOWR
	127	126	122	120	125	121	65	69	64	66	70	71	124
	Z	X	C	V	B	N	M	,	.	/			Atari logo
	87	86	82	80	85	99	101	96	98	102	103		
SPACE BAR		F1	F2	F3	F4	HELP							
97		67	68	83	84	81							

Keyboard Values with CONTROL*

ESC	1	2	3	4	5	6	7	8	9	0	<	>	DEL
156	159	158	154	152	157	155	179	181	176	178	182	183	180
TAB	Q	W	E	R	T	Y	U	I	O	P	-	=	RETURN
172	175	174	170	168	173	171	139	141	136	138	142	143	140
	A	S	D	F	G	H	J	K	L	;	+	*	CAPS/LOWR
	191	190	186	184	189	185	129	133	128	130	134	135	188
	Z	X	C	V	B	N	M	,	.	/			Atari logo
	151	150	146	144	149	163	165	160	162	166	167		
SPACE BAR		F1	F2	F3	F4	HELP							
161		131	132	147	148	145							

Keyboard Values with SHIFT and CONTROL

ESC	1	2	3	4	5	6	7	8	9	0	<	>	DEL
220	223	222	218	216	221	219	243	245	240	242	246	247	244
TAB	Q	W	E	R	T	Y	U	I	O	P	-	=	RETURN
236	239	238	234	232	237	235	203	205	200	202	206	207	204
	A	S	D	F	G	H	J	K	L	;	+	*	CAPS/LOWR
	255	254	250	248	253	249							252
	Z	X	C	V	B	N	M	,	.	/			Atari logo
						227	229	224	226	230	231		
SPACE BAR		F1	F2	F3	F4	HELP							
225													

*Eleven keys cannot be read with SHIFT and CONTROL pressed: J, K, L, ;, +, *, Z, X, C, V, and B (and F1, F2, F3, F4, and HELP on XL models).

the latest key pressed or executes an INPUT statement, it will think the key you specified was pressed.

You can also change the way the computer thinks the keys are laid out. For instance, you might want to try the Dvorak keyboard. The Qwerty keyboard (the one your computer comes with) was deliberately designed to be inconvenient and slow. Back when mechanical typewriters were first used, quick typists kept jamming the keys. So the Qwerty keyboard puts the most commonly used letters off the home keys or on the left side, where most typists will have a harder time getting to them. Computerized keyboards are faster now, and the Dvorak keyboard is designed to take advantage of that. The most commonly used characters are on the home keys. And you can learn the Dvorak system by making your Atari read the keyboard in the Dvorak pattern, just by reconfiguring the relationship between KEYCODE and ATASCII.

You can also use the keyboard codes to get input from the keyboard directly, bypassing the OS's formulas for conversion. That's the use we'll pursue in the rest of this article.

Exceptions to the Rules

SHIFT-lock. It is important to remember that the number stored at 53769 and shadowed at 764 is the value of the key combination actually pressed. It is not affected at all by whether the keyboard is SHIFT-locked or CONTROL-locked.

When the Atari powers up, the keyboard is locked into the alphabetic shift mode—when you press any letter key, with or without pressing SHIFT at the same time, the shifted value appears on the screen. But as far as locations 53769 and 764 are concerned, if you don't press SHIFT, the unshifted value is all it gets.

The way the operating system handles SHIFT-lock and CONTROL-lock is simple—you can imitate this in your own programs. When the CAPS/LOWR key is pressed, the operating system changes the SHIFT-lock flag at location 702. If the CAPS/LOWR key is pressed by itself, 0 is stored at 702; if SHIFT and CAPS/LOWR are pressed together, 64 is stored there; and if CONTROL and CAPS/LOWR are pressed together, 128 is stored there. From then on, if the key pressed calls for an alphabetic (letter, rather than number or symbol) character, the operating system checks location 702 and adds the number stored there to the offset into the Key Definition Table. Programs 1 and 2 both

perform a customized version of this function, by-passing the operating system entirely.

XL models. XL models (Atari 600XL, 800XL, 1200XL, 1400XL, and 1450XL) allow you to lay out your own Keycode Definitions Table (essentially what the programs in this article do with the array AC (*n*)), and inform the operating system by POKEing the address of the table, low byte first, into locations 121 and 122 (\$79 and \$7A). The table is set up exactly like the ATASCII array—you could use the DATA statements, converting them from ICODE to ATASCII order, to set up the table for the XL redefinition.

The XL models also allow you to redefine the *F_n* and SHIFT-*F_n* keys separately, without redefining the entire keyboard, by setting up an eight-byte table and POKEing its address, low byte first, into locations 96 and 97 (\$60 and \$61).

However, this system of keyboard redefinition still leaves you with the OS's system of interpretation, which ignores all SHIFT-CONTROL and all CONTROL-*number key* combinations. To really take advantage of the power of the keyboard code, you need to set up your own interpretation system as well.

Missing SHIFT-CONTROL combinations. Eleven keys—16 on XL models—return no value to location 53769 if both SHIFT and CONTROL are pressed at the same time: J, K, L, ;, +, *, Z, X, C, V, and B on all Ataris, and F1, F2, F3, F4, and HELP on XL models. It is as if those key combinations did not exist.

Interrupts. Most of the time, whatever number is stored in location 53769 is also stored at 764. There are exceptions when the key combination is acted on during an interrupt. The CONTROL-1 combination, for instance, is read during an interrupt and can't be read from 764—but the value still occurs at location 53769 and can be read there. On XL models, CONTROL-F1, CONTROL-F2, CONTROL-F4, and HELP, SHIFT-HELP, and CONTROL-HELP also generate codes that are not transferred from 53769 to 764.

What difference does this make? If you want to be able to read that code in spite of the interrupt, you can—by reading 53769 instead of 764. The interrupt will still take place, but your program will also “know” that the key combination was pressed. Or if you want your program to ignore keys used by the interrupts, read the values at 764 instead of 53769.

Here is a short program that reads the hardware register and POKEs the raw KEYCODE number into screen memory. First, you will see that the KEYCODE number has no relation to the

ICODE number that normally is POKEd into screen memory.

Second, since the PRINT command isn't being used, the CONTROL-1 key has no effect at all—and its KEYCODE number is POKEd into screen memory, where it appears as an inverse question mark. If you have an XL model, you will see that pressing CONTROL-F4 still toggles between the standard and international character sets—but it also causes an inverse 4 to appear on the screen.

```
10 POKE PEEK(88)+256*PEEK(89)+N, PEEK(53769)
20 N=N+1-960*(N>958):GOTO 10
```

Built-in delay. There is a slight but measurable time lag between the keypress causing a number to be stored at 53769, and the echo getting stored in 764. Here is a very short example program that will show you these codes:

```
10 PRINT PEEK(764);" ";PEEK(53769)
20 FOR I=0 TO 40:NEXT I:GOTO 10
```

This program PRINTs the value at 764 on the left and the value at 53769 on the right. If you RUN this program and then type very quickly, you will sometimes see a number appear on the right that has not yet appeared on the left—you have caught the OS between receiving the KEYCODE at 53769 and echoing it at 764. If your program needs speed (particularly if it is a machine language routine), you'll definitely want to read the keyboard code at 53769.

ATASCII-ICODE Conversions

Actually, since ATASCII and ICODE have a regular relationship, conversions back and forth are quite simple. Subroutine 1 converts the ATASCII number AC(N) to ICODE and assigns the value to IC(N):

Subroutine 1. ATASCII to ICODE

```
800 VERS=0: IF AC>127 THEN VERS=1: AC=AC-128
810 IF AC<32 THEN IC=AC+64+128*VERS: RETURN
820 IF AC<96 THEN IC=AC-32+128*VERS: RETURN
830 IC=AC+128*IV: RETURN
```

When you jump to this subroutine, the variable AC must contain the ATASCII value of the character you want converted to ICODE. When you return from the subroutine, the variable IC will contain the ICODE value. You can POKE it to screen memory: POKE PEEK(88)+256*PEEK(89)+OFFSET,IC.

Subroutine 2 converts from ICODE to ATASCII:

Subroutine 2. ICODE to ATASCII

```
200 IV=0: IF IC>127 THEN IV=1: IC=IC-128
210 IF IC<64 THEN AC=IC+32+128*IV: RETURN
220 IF IC<96 THEN AC=IC-64+128*IV: RETURN
230 AC=IC+128*IV: RETURN
```

When you GOSUB to this routine, the variable IC contains the ICODE value of the character you want converted. When you return from the subroutine, the variable AC will contain the value that, when PRINTed, will cause the character to be displayed.

In both subroutines, the variable IV is used to keep track of whether the character was inverse or not. Note that you cannot change the order of these lines. If 220 is executed before 210, or 120 before 110, the results will be wrong.

The Keyboard Code Array

Since KEYCODE doesn't have a systematic relationship with the other codes, a simple program wouldn't convert to and from KEYCODE. A much better solution is to set up a table of ATASCII or ICODE values in KEYCODE order, and then use the KEYCODE number as a pointer into the table to find the right ATASCII or ICODE value. In BASIC, the simplest way of doing this is to use the KEYCODE number as the subscript in an array containing either ICODE or ATASCII values. (For a complete listing of keyboard codes and their relationship to internal code and ATASCII, see Appendix A, "A Complete Guide to the Atari Character Set.")

During your program's setup phase, you need to DIMension one or both of these arrays:

```
DIM IC(255), AC(255)
```

The elements of this array will be assigned either ATASCII or ICODE values, arranged in KEYCODE order. For instance, KEYCODE 0 is produced by pressing l (lowercase L). Therefore, the value of C(0) will be 108.

Once the array has been set up, the keyboard can be read almost instantly. For instance, to PRINT the last key pressed, regardless of what it was or how long ago it was pressed, this statement would do:

```
PRINT AC(PEEK(764))
```

We'll go into much more detail about effective use of the keyboard codes later on.

Assigning values. The way you assign values to this array depends on how you want to use the keyboard data.

KEYCODE 94 is produced by pressing SHIFT-2 (the quotation mark). If the array has been set up in ATASCII order, the value of C(94) will be 34. This is the method you will use if you want to PRINT CHR\$(C(KEYCODE)) or create strings.

If the array has been set up in ICODE order, the value of C(94) will be 2. This is the method you will use if you want to POKE keyboard input directly into screen memory, to create displays without using PRINT or strings.

The KEYCODE DATA Statements

Program 1 is the heart of this system. It consists of DATA statements that contain ICODE values in KEYCODE order.

Extra key combinations. Zero is used for every KEYCODE value that has not been assigned an ATASCII or ICODE value. There are many zeros in the DATA statements, even though only the space bar should produce a blank, because there are many KEYCODE values that have no corresponding ICODE or ATASCII values. For instance, SHIFT-RETURN has no special ICODE or ATASCII value. If you wanted SHIFT-RETURN to have the same value as RETURN, you would assign it the same value as RETURN.

Inverse ATASCII characters. Some ATASCII values are really inverse characters. ATASCII 156-159 (usually produced by pressing SHIFT-DELETE, SHIFT-INSERT, CONTROL-TAB, and SHIFT-TAB) PRINT as nothing more than the inverse of ATASCII 28-31. ATASCII 253, 254, and 255 (CONTROL-2, CONTROL-DELETE, and CONTROL-INSERT) are inverses of ATASCII 125-127 (SHIFT-CLEAR, DELETE, and TAB). ATASCII 155 (RETURN), if it could be PRINTed as a character, would be the inverse of ATASCII 27(ESC). Since all these characters can be obtained by PRINTing an inverse of another key combination, they have been left as zeros in the DATA statements. (If you want a keyboard code to clear the screen or ring the CONTROL-2 buzzer, that can be done independently, as will be shown below.)

Impossible codes. Many of the zeros in the DATA statements are there because certain KEYCODE values cannot exist—no combination of keys will result in that particular number. The impossible codes on non-XL Ataris are 3, 4, 9, 17, 19, 20, 25, 36, 41, 49, and 59—and those numbers plus 64, 128, and 192. Since the DATA statements are arranged in KEYCODE order, the

impossible codes are represented by zeros just to keep the array in order. If your computer is an XL model, 3, 4, 17, 19, and 20 represent F1, F2, HELP, F3, and F4, and can be read in any combination except SHIFT-CONTROL.

Assigning Values to the Array

Program 1 includes all the DATA statements needed to set up arrays AC(*n*) and IC(*n*). By removing the word REM in front of the subroutine calls, you can create a disk file containing the array, or load the data from the disk file into the array. Or you can simply add these DATA statements to a program.

Using the Keyboard Code in a Program

Once the array is set up, reading the keyboard code is very simple. You can use it directly, of course, by putting it in a function:

```
PRINT CHR$(PEEK(53769))
```

However, this does not begin to use the freedom the keyboard code gives you.

Is a key pressed? First, if your keyboard read routine is complicated at all, you will want to avoid going through it when there is nothing to read. In the main loop of your program, the test can be as simple as this:

```
ON PEEK(753)<>3 GOSUB 500
```

Location 753 is set to 3 every time a key is pressed. If a key is not pressed, it decrements (decreases in value by 1) every 1/60 second until it reaches zero. If a key is pressed and held down, 753 will continue to equal 3. So your program will GOSUB to your keyboard read routine only when a key is pressed.

Locking character sets. The CAPS-LOWR key usually affects only the alphabetic character keys. To get the % character, you have to press SHIFT-5, regardless of whether the alphabetic keys are locked in SHIFT or CONTROL mode. The subroutine below, however, will automatically lock *all* the keys in one mode or another.

```
2500 N=PEEK(53769):S=INT(N/64):KEY=N-S*64
2510 IF KEY=60 THEN SHIFT=S*64:RETURN
2530 IC=IC(KEY+SHIFT):AC=AC(KEY+SHIFT)
2540 POKE W,IC:W=W+1-960*(W=959):RETURN
```

Line 2500 sets up three useful variables. N holds whatever value

was in 53769. S tells us, in effect, whether SHIFT, CONTROL, or both were also pressed. If S = 0, then neither was depressed; if S = 1, then SHIFT; if S = 2, then CONTROL; if S = 3, then both. KEY tells us which actual key was depressed, regardless of whether SHIFT or CONTROL was depressed.

In line 2510 the program determines whether KEY was the CAPS-LOWR key, whose code is 60. If it was, then the variable SHIFT is set at 0, 64, 128, or 192, depending on whether SHIFT or CONTROL was depressed. Since CAPS-LOWR is not a printing character, the subroutine returns at this point.

In line 2530, IC and AC are set at the ICODE and ATASCII equivalent, not of KEY, but of KEY + SHIFT. Whatever value SHIFT was last given by line 2510 is automatically added to the absolute value of whatever key was depressed. Now if the program should print AC or POKE IC onto the screen, it would give either its shifted, control, or unshifted value, depending on the value of SHIFT, regardless of whether SHIFT or CONTROL was pressed when the key was entered.

In line 2540, IC is POKEd into location SC + W, which represents a position in screen memory. (SC = lowest address of screen memory; W = current location above SC.) Then W is incremented (increased by 1). If W is at 959, so that incrementing it would take us off the bottom of the screen, the program subtracts 960 and starts us at the upper left-hand corner again.

Inverse mode. Right now there's no way to print inverse characters. So let's add a line to take care of that.

```
2505 IF KEY=39 AND S>0 AND S<3 THEN IV=128*(S=2):
RETURN
```

We also need to change line 2530:

```
2530 IC=IC(KEY+SHIFT+IV):AC=AC(KEY+SHIFT+IV)
```

Now when you press the Atari logo key at the same time you press CONTROL, the entire keyboard shifts into inverse mode. Press SHIFT and the Atari logo key and the keyboard shifts back into regular mode. But when you press the Atari logo key by itself or with *both* CONTROL and SHIFT, there is no effect on inverse mode at all.

Multiple meanings. When you press the arrow keys when the keyboard is locked into the control mode, you'll notice that the arrows appear on the screen, and the cursor does not move. This is because the program is POKing the ICODE values into screen

memory. If the program were PRINTing the ATASCII values, the cursor would have moved.

But you can still use the cursor keys, just as you always have, along with the SHIFT-CLEAR key, by adding these lines:

```
2520 ON S GOTO 2800,2850,2900
2800 IF N=125 THEN PRINT CHR$(AC(N))
2810 RETURN
2850 H=(KEY=7)-(KEY=8):V=40*((KEY=14)-(KEY=15)):
W=W+H+V
2860 IF W<0 THEN W=W+960:RETURN
2870 IF W>959 THEN W=W-960
2880 RETURN
2900 REM This command line is executed if
SHIFT-CONTROL are pressed
2910 RETURN
```

Notice that in line 2520 the program uses a GOTO instead of a GOSUB. This means that the RETURN at the end of each of these subroutines will take us back, not to the statement immediately following the branch in line 2520, but to the main loop of the program. If we did not do this, every command would also result in a blank being displayed on the screen.

More Commands Than You Can Use

Remember when I said that we would have 68 command characters? Now you can see that we could just as easily have 140 command characters. That is because, by using the CAPS-LOWR key the way we do, all the printable values of each key can be displayed on the screen *without* pressing SHIFT or CONTROL each time. Then if the user *does* press SHIFT or CONTROL or both with a character, we can interpret that separately as a command.

Naturally, few programs would ever need 140 command characters. And a word processing program would do much better to interpret keys pressed with SHIFT as characters rather than commands—typists would hate having to use CAPS-LOWR every time they wanted a capital letter or a shifted symbol.

But using the keyboard codes, you have the freedom to design your own keyboard system, to respond to the exact needs of your own program. You could design a word processor that used a keyboard layout different from the standard Qwerty, or you could simply speed up the key repeat. You could also use a section of the keyboard as a game controller with continuous commands—as long as a key was held down, it would continue to repeat its function. You could read the keyboard as an organ,

shifting back and forth between different banks of keys with different stops set.

The Dvorak Keyboard

One thing you might want to try is the Dvorak keyboard (Program 2). In this program, the DATA sets up the arrays so the keyboard is interpreted according to the Dvorak keyboard instead of the Qwerty keyboard (see Table 2). By using this table with your own keyboard reading program, you could train yourself to type with the much faster Dvorak keyboard arrangement.

Table 2. Dvorak Keyboard Codes

Unshifted Keyboard Values													
ESC	1	2	3	4	5	6	7	8	9	0	<	>	DEL
28	31	30	26	24	29	27	51	53	48	50	54	55	52
TAB /				P	Y	F	G	C	R	L	-	=	RETURN
44	47	46	42	40	45	43	11	13	8	10	14	15	12
	A	O	E	U	I	D	H	T	N	S	+	*	CAPS/LOWR
	63	62	58	56	61	57	1	5	0	2	6	7	60
	;	Q	J	K	X	B	M	W	V	Z	Atari logo		
	23	22	18	16	21	35	37	32	34	38	39		
SPACE BAR	F1	F2	F3	F4	HELP								
33	3	4	19	20	17								

Keyboard Values with SHIFT													
ESC	1	2	3	4	5	6	7	8	9	0	<	>	DEL
28	31	30	26	24	29	27	51	53	48	50	54	55	52
TAB ?	[]		P	Y	F	G	C	R	L	-	=	RETURN
44	47	46	42	40	45	43	11	13	8	10	14	15	12
	A	O	E	U	I	D	H	T	N	S	+	*	CAPS/LOWR
	63	62	58	56	61	57	1	5	0	2	6	7	60
	:	Q	J	K	X	B	M	W	V	Z	Atari logo		
	23	22	18	16	21	35	37	32	34	38	39		
SPACE BAR	F1	F2	F3	F4	HELP								
33	3	4	19	20	17								

Keyboard Values with CONTROL

See Table 1.

Keyboard Values with SHIFT and CONTROL

See Table 1.

Note: The Dvorak keyboard calls for the single and double quotation marks to be just to the right of the L key and the hyphen and underline characters to be just to the right of the S key. The preceding table does not show this because those keys are used for arithmetic functions on the Atari keyboard, and most users would probably prefer to leave those keys as they are.

Debounce routine. Line 100 contains a homemade debounce routine. When you type, your finger remains on the key for a fraction of a second. If the program reads the keyboard again before you lift your finger, the key will repeat—even though you might not want it to. The debounce routine checks to see if the value it just got from the keyboard is the same as the last one it got. If not, a new key has been pressed and the program goes on. But if the keys are the same, the counter *X* is incremented by one. If *X* is less than 4, the key will be ignored; if it is greater than 4, it is assumed that the typist meant the key to repeat.

By changing the 4 to some other number, you can change the time lag between holding down a key and getting it to repeat on the screen. Or you could write a routine that would cause the cursor control keys to repeat without a much shorter debounce delay than the other keys.

Because this routine is written in BASIC, it has another problem—it's possible for you to type so quickly that you press one key and then go on and press another key before the program ever reads the first key's value. You can solve the problem by writing in machine language. Or you could write just your keyboard reading routine in machine language and run it in an interrupt, have that routine store the characters typed into a buffer, and let your BASIC program read the keyboard input from the buffer at its own speed. Or you could compile your BASIC program so it ran faster than people could type. But the more commands you have to check for with each letter typed, the slower your BASIC program will run, and the more keystrokes you'll lose because of slow program execution.

SHIFting. This program improves on the way Program 1 handles the SHIFT key. Instead of simply ignoring the SHIFT and CONTROL keys except when CAPS/LOWR is pressed, Program 2 pays attention to SHIFT. If the keyboard is locked into SHIFT or CONTROL, pressing the SHIFT key has no effect. If the keyboard is locked into lowercase (that is, if you pressed CAPS/LOWR by itself), then pressing the SHIFT key with another key will cause that letter, and only that letter, to be shifted—just like the standard typewriter keyboard.

This is handled in line 105, when *S* is set to equal $\text{INT}(K/64)$. In effect, this makes *S* equal 1 if SHIFT is pressed, 2 if CONTROL is pressed, 3 if both are pressed, and 0 if neither is pressed. Then, in line 120, *N* is set back to the value of *K*, the original keystroke

combination, if SHIFT was pressed. This by-passes the locked value of the variable SHIFT for one keystroke only.

It would be a simple matter to adapt this program so that if the keyboard is locked into SHIFTed condition, pressing the SHIFT key and another key would cause the program to display the lowercase, unshifted value of that key. Or you could write a routine that would allow you to lock and unlock the number keys into shifted and unshifted condition separately from the rest of the keyboard.

POKEing to the screen. This program pretends to be a typing program, since lines 200 and 205 POKE the letters directly into screen memory. Each time a character is POKEd into memory, the pointer variable E is incremented by one so that the next character will be placed just to the right of the character before.

An alternative would be to replace ICODE screen POKEing with ATASCII PRINT statements. Delete lines 200 and 205 and replace them with

```
200 PRINT CHR$(AC(N))+VERS
```

Now the editing functions will work and the screen will scroll when you reach the bottom.

These programs, while not especially useful in themselves, should give you a pretty good idea of some of the possibilities that are opened up to you if your programs read the keyboard directly. Whenever you write a program that relies heavily on keyboard input, you should give serious consideration to having your program read the keyboard independently—it might allow you to add refinements to your program that make it more powerful or useful to the user.

Program 1. Standard Array

```
5 DIM IC(255),AC(255):SC=PEEK(88)+256*PEEK(89):SHIFT=64:VERS=0
10 GOSUB 500:REM THIS WILL CREATE ARRAYS FROM DATA STATEMENTS
15 REM GOSUB 600:REM USE THIS TO CREATE "D:KEYCODE.DAT"
20 REM GOSUB 700:REM USE THIS TO CREATE ARRAYS FROM DISKFILE "D:KEYCODE.DAT"
100 POKE 694,0:ON PEEK(753)<>3 GOTO 100:K=PEEK(764)
```

```
105 N=K: IF N>63 THEN N=N-64: IF N>63 THEN N=N
-64: IF N>63 THEN N=N-64
110 IF N=60 THEN SHIFT=4+K-64
115 IF N=39 THEN VERS=128*(VERS<>128)
200 PRINT N, CHR$(AC(N+SHIFT)+VERS)
205 POKE SC+959, IC(N+SHIFT)+VERS
210 GOTO 100
500 RESTORE 1000: FOR I=0 TO 191: READ N: IC(I)
=N: NEXT I: FOR I=192 TO 255: IC(I)=0: NEXT I
510 FOR I=0 TO 255: N=IC(I)
520 IF N<64 THEN AC(I)=N+32
530 IF N>63 AND N<96 THEN AC(I)=N-64
540 IF N>95 THEN AC(I)=N
550 NEXT I
560 RETURN
600 OPEN #4,8,0,"D:KEYCODE.DAT"
605 RESTORE 1000: FOR I=0 TO 191: READ N: PUT #
4,N: NEXT I
610 FOR I=192 TO 255: PUT #4,0: NEXT I
615 CLOSE #4: RETURN
700 OPEN #4,4,0,"D:KEYCODE.DAT"
705 FOR I=0 TO 255: GET #4,N: IC(I)=N
710 IF N<64 THEN AC(I)=N+32
715 IF N>63 AND N<96 THEN AC(I)=N-64
720 IF N>95 THEN AC(I)=N
725 NEXT I: RETURN
1000 DATA 108,106,27,0,0,107,11,10,111,0,112
,117,0,105,13,29
1016 DATA 118,0,99,0,0,98,120,122,20,0,19,22
,91,21,18,17
1032 DATA 12,0,14,110,0,109,15,0,114,0,101,1
21,127,116,119,113
1048 DATA 25,0,16,23,126,24,28,30,102,104,10
0,0,0,103,115,97
1064 DATA 44,42,26,0,0,43,60,62,47,0,48,53,0
,41,63,124
1080 DATA 54,0,35,0,0,34,56,58,4,0,3,6,0,5,2
,1
1096 DATA 59,0,61,46,0,45,31,0,50,0,37,57,0,
52,55,49
1112 DATA 8,0,9,7,0,32,125,0,38,40,36,0,0,39
,51,33
1128 DATA 76,74,123,0,0,75,94,95,79,0,80,85,
0,73,92,93
1144 DATA 86,0,67,0,0,66,88,90,0,0,0,0,0,0,0
,0
1160 DATA 64,0,96,78,0,77,0,0,82,0,69,89,0,8
4,87,81
1176 DATA 0,0,0,0,0,0,0,0,70,72,68,0,0,71,83
,65
```

Program 2. Dvorak Array

```

5 DIM IC(255),AC(255):SHIFT=64:VERS=0:SC=PEEK
  K(88)+256*PEEK(89):E=0
10 GOSUB 500:REM THIS WILL CREATE ARRAYS FROM
DATA STATEMENTS
15 REM GOSUB 600:REM USE THIS TO CREATE "D:D
VORAK.DAT"
20 REM GOSUB 700:REM USE THIS TO CREATE ARRAYS
FROM DISKFILE "D:DVORAK.DAT"
100 POKE 694,0:ON PEEK(753)<>3 GOTO 100:P=PEEK
  (53769):IF P=K THEN X=X+1:IF X<4 THEN
  100
105 X=0:K=P:S=INT(K/64):N=K-64*S
110 IF N=60 THEN SHIFT=64*S
115 IF N=39 THEN VERS=128*(VERS<>128)
120 N=N+SHIFT:IF S=1 THEN N=K
200 POKE SC+E,IC(N)+VERS
205 E=E+1-960*(E>958)
210 GOTO 100
500 RESTORE 1000:FOR I=0 TO 191:READ N:IC(I)
  =N:NEXT I:FOR I=192 TO 255:IC(I)=0:NEXT
  I
510 FOR I=0 TO 255:N=IC(I)
520 IF N<64 THEN AC(I)=N+32
530 IF N>63 AND N<96 THEN AC(I)=N-64
540 IF N>95 THEN AC(I)=N
550 NEXT I
560 RETURN
600 OPEN #4,8,0,"D:KEYCODE.DAT"
605 RESTORE 1000:FOR I=0 TO 191:READ N:PUT #
  4,N:NEXT I
610 FOR I=192 TO 255:PUT #4,0:NEXT I
615 CLOSE #4:RETURN
700 OPEN #4,4,0,"D:KEYCODE.DAT"
705 FOR I=0 TO 255:GET #4,N:IC(I)=N
710 IF N<64 THEN AC(I)=N+32
715 IF N>63 AND N<96 THEN AC(I)=N-64
720 IF N>95 THEN AC(I)=N
725 NEXT I:RETURN
1000 DATA 110,104,115,0,0,116,11,10
1008 DATA 114,0,108,103,0,99,13,29
1016 DATA 107,0,106,0,0,120,113,27
1024 DATA 20,0,19,22,91,21,18,17
1032 DATA 119,0,118,98,0,109,122,0
1040 DATA 112,0,14,102,127,121,12,15
1048 DATA 25,0,16,23,126,24,28,30
1056 DATA 117,100,101,0,0,105,111,97
1064 DATA 46,40,51,0,0,52,60,62

```

```
1072 DATA 50,0,44,39,0,35,63,124
1080 DATA 43,0,42,0,0,56,49,26
1088 DATA 4,0,3,6,0,5,2,1
1096 DATA 55,0,54,34,0,45,58,0
1104 DATA 48,0,61,38,0,57,59,31
1112 DATA 8,0,9,7,0,32,125,0
1120 DATA 53,36,37,0,0,41,47,33
1128 DATA 76,74,123,0,0,75,94,95
1136 DATA 79,0,80,85,0,73,92,93
1144 DATA 86,0,67,0,0,66,88,90
1152 DATA 0,0,0,0,0,0,0,0
1160 DATA 64,0,96,78,0,77,0,0
1168 DATA 82,0,69,89,0,84,87,81
1176 DATA 0,0,0,0,0,0,0,0
1184 DATA 70,72,68,0,0,71,83,65
```

Using the Atari Timer

Stephen Levy

Because FOR/NEXT loops are not accurate timers, the solution is to incorporate Atari's internal counters into programs where you want something delayed or timed reliably.

Have you ever written a program and wanted a specific time delay? What did you do? Some of us figured a FOR/NEXT loop was the answer, so we set to work with our stopwatches until we found that the following takes about three seconds to write "STOP":

```
10 PRINT "BEGIN"  
20 FOR X=1 TO 1000  
30 NEXT X  
40 PRINT "STOP"
```

Then we went along and wrote our programs and found that our three-second delay had become five, six, or even ten seconds. Why? Because the Atari FOR/NEXT loops take longer as you add lines of code to the program.

There is a better way. Yes, machine language routines are great for timing on the Atari, especially if you know how to use locations 536 to 558 (\$218 to \$22E). But it can be most disconcerting if you allow some of those registers to drop to zero unchecked.

Accurate Delays

BASIC programmers, there is a way. Use memory locations 18, 19, and 20.

These timers work like the mileage gauge on a car's speedometer: one counter counts up and then sets the one next to it which, in turn, sets the next one. Each counter on the speedometer goes up when the one to its right hits ten. In the computer, they count up to 255 before going back to zero.

Register number 20 counts at the rate of 60 numbers per second up to number 255, then increments register 19 by one and

starts over. When register 19 reaches 255, it increments register 18 by one. If you POKE zero into all three registers, it will take about 1092 seconds before a one appears in register 18 (more than 18 minutes). The table gives some times (it assumes all three registers began with zero). Notice that it would take more than 77 hours for memory location 18 to reach 255.

Well, how does all this help? Let's look at our short program again. We can rewrite it this way:

```
10 PRINT "BEGIN": POKE 20,0
20 IF PEEK(20)<180 THEN 20
30 PRINT "STOP"
```

This routine will continue to take three seconds no matter how long your program. Well, not exactly; since it is written in BASIC, the longer the program, the longer the routine will take. But the influence of the program length will usually be negligible.

Included here are three programs which demonstrate a much more functional use of this timer. Type in Program 1, leaving out the REM statements. This program tells the user the time interval between the pressing of RETURN after typing RUN and the pressing of RETURN a second time. Notice that if you press another key the computer goes back to line 140.

This short program demonstrates several useful concepts. First, the computer is looking for a particular input, in this case the RETURN key (ATASCII 155). Second, line 160 PEEKs at registers 18, 19, and 20. Notice we POKEd location 20 last on line 130 and PEEKed at it first on line 160. Third, line 170 contains the important formula for converting the information in locations 18, 19, and 20 to seconds. Why 4.267? Because 256 divided by 60 numbers per second equals 4.267. Fourth, lines 180 to 200 convert the total number of seconds to minutes and seconds.

Program 2 is a bit more useful. It is a timed math quiz in which the user is allowed eight and one-half seconds to answer. Line 140 is used to check if a key has been pressed. If no key has been pressed, then the program goes back to check how much time has elapsed. Once a key is pressed, the computer GETs the ATASCII code and calls it A1. At lines 160 and 170, A1 is converted to its CHR\$ and placed in its proper place in ANS\$. If A1 equals 155 (ATASCII code for the RETURN key), the program moves to line 220, where the value of ANS\$ is put into variable ANS.

The final illustration, Program 3, is also a math quiz. In this

case the user is given unlimited time. This program combines elements of both Programs 1 and 2.

This Atari timing device should be beneficial whether you wish to impose a time limit, simply time answers, or have users compete against each other or themselves. The timer has applications for both educational programming and games. With some experimentation you should be able to adapt this timing device for use with your own programs.

Sample Times

LOC.20	LOC.19	LOC.18	TIME MIN:SEC
60	0	0	0:01
60	1	0	0:05
0	2	0	0:08
100	2	0	0:10
0	3	0	0:12
100	4	0	0:18
21	14	0	1:00
42	28	0	2:00
84	56	0	4:00
176	112	0	8:00
0	255	0	18:08
0	60	2	40:40
0	0	16	291:17
0	0	100	1820:35
0	0	150	2730:52
0	0	255	4642:29

Program 1. Atari Timer

```

10 REM ATARI TIMER
20 REM
30 REM THIS PROGRAM DEMONSTRATES HOW
40 REM TO USE ATARI TIMER:
50 REM ADDRESS 18,19,20
60 REM IT FIGURES HOW LONG IT TAKES
70 REM YOU TO PRESS THE <RETURN> KEY.
80 REM RUN THE PROGRAM THEN PRESS
90 REM <RETURN>
100 REM PROGRAM RUNS BETTER WITHOUT
110 REM REMARK STATEMENTS OR GOTO 120
120 OPEN #1,4,0,"K:"
130 FOR Z=18 TO 20:POKE Z,0:NEXT Z
140 GET #1,D:IF D=155 THEN 160

```



```

150 GOTO 140
160 A=PEEK(20):B=PEEK(19):C=PEEK(18)
170 SEC=INT((4.267*256*C)+(B*4.267)+(A/60))
180 MIN=INT(SEC/60)
190 M=MIN*60
200 SEC=SEC-M
210 PRINT MIN;" MINUTES ";SEC;" SECONDS"

```

Program 2. Timed Math Quiz

```

10 REM TIMED MATH QUIZ
20 REM
30 REM THIS IS A TIMED MATH QUIZ
40 REM CHANGE LINE 130 TO A=1
50 REM ALLOWS 4 1/4 SECOND
60 REM A=2 ALLOWS 8 1/2 SECONDS
70 REM A=3 ALLOWS 12 3/4 SECONDS, ETC.
80 OPEN #1,4,0,"K:":DIM ANS$(10)
90 PRINT :Q1=INT(RND(0)*20):Q2=INT(RND(0)*20)
   ):X=1
100 PRINT Q1;" + ";Q2;"=";
110 POKE 18,0:POKE 19,0:POKE 20,0
120 A=PEEK(19):B=PEEK(20)
130 IF A=2 THEN 180:REM 8 1/2 SECONDS
140 IF PEEK(764)=255 THEN 120
150 GET #1,A1:IF A1=155 THEN 220
160 ANS$(X,X)=CHR$(A1)
170 PRINT ANS$(X,X);:X=X+1:GOTO 120
180 PRINT :PRINT "TIME'S UP"
190 PRINT "THE ANSWER IS ";Q1+Q2
200 FOR W=1 TO 400:NEXT W
210 ANS$=" ":GOTO 90
220 ANS=VAL(ANS$):PRINT
230 IF ANS=Q1+Q2 THEN PRINT :PRINT "CORRECT"
   ):GOTO 200
240 PRINT :PRINT "SORRY":PRINT :GOTO 190

```

Program 3. Revised Math Quiz

```

10 REM REVISED MATH QUIZ
20 REM
30 REM THIS PROGRAM COMBINES ELEMENTS
40 REM OF PROGRAMS 1 AND 2.
50 REM IT GIVES MATH QUIZ AND TELL HOW
60 REM LONG IT TOOK YOU TO DO EACH
70 REM PROBLEM.
80 OPEN #1,4,0,"K:":DIM ANS$(10)
90 PRINT :Q1=INT(RND(0)*20):Q2=INT(RND(0)*20)
   ):X=1

```

```
100 PRINT Q1;" + ";Q2;"=";
110 POKE 18,0:POKE 19,0:POKE 20,0
120 IF PEEK(764)=255 THEN 120
130 GET #1,A1:IF A1=155 THEN 190
140 ANS$(X,X)=CHR$(A1)
150 PRINT ANS$(X,X);:X=X+1:GOTO 120
160 PRINT "THE ANSWER IS ";Q1+Q2
170 FOR W=1 TO 1000:NEXT W
180 ANS$=" ":GOTO 90
190 A=PEEK(20):B=PEEK(19):C=PEEK(18)
200 ANS=VAL(ANS$):PRINT
210 IF ANS=Q1+Q2 THEN PRINT :PRINT "CORRECT"
    :GOTO 230
220 PRINT :PRINT "SORRY"
230 SEC=INT((4.25*256*C)+(B*4.25)+(A/60))
240 MIN=INT(SEC/60)
250 M=MIN*60
260 SEC=SEC-M
270 IF MIN<>0 THEN 290
280 PRINT "THAT TOOK YOU ";SEC;" SECONDS":GO
    TO 300
290 PRINT "THAT TOOK YOU ";MIN;" MINUTES":PR
    INT "AND ";SEC;" SECONDS"
300 GOTO 170
```

Blinking Characters

Frank C. Jones

Make your messages stand out by having them blink. The technique is easy and simple to add to your programs. Once the machine language routine is POKEd into memory, the BASIC program can be removed—leaving the machine language there to do the work necessary for “Blinking Characters.”

The inverse video key on the Atari computer allows messages to be displayed in inverse video for special emphasis or eye-catching effects. Another, sometimes even more dramatic, method of catching the viewer's eye is to have the message flash on and off, or blink. There is no simple command in Atari BASIC to produce this effect, but the key to producing it lies in the register, maintained by the operating system, called CHACT, decimal address 755 (\$2F3). If bit one in this register is set to one, inverse video characters are displayed in inverse video; if it is set to zero, they are displayed normally. However, if bit zero is set to one, these characters are displayed as blank spaces (inverse video or normal blanks depending on bit one).

Look for a Faster Solution

With this information we can immediately write a program that will produce blinking characters on the screen, as Program 1 does. The trouble with this approach is that our BASIC program is completely preoccupied with timing loops and toggling bit zero of CHACT. If we try to incorporate this routine in a program that does anything else, the timing gets very difficult if not downright impossible. What we really want is a routine that will sit in the background and toggle bit zero of CHACT on a regular basis without interfering with any BASIC program that might be running at the time. Fortunately, the Atari has in it the resources we need to do just this.

The Atari operating system maintains five separate timers that are incremented or decremented during every vertical blank period (the period between successive TV picture frames during which the screen is dark). Actually, most of them are updated only during “second stage” vertical blank; more about this in a

moment. One of these, called CDTMV2 (\$21A), is a two-byte down counter that can be set to any value between 1 and 65535. Every sixtieth of a second, during vertical blank, the operating system reduces this number by one, and when it counts to zero it performs a subroutine jump to the address that it finds in the two-byte vector called CDTMA2 (\$228) and returns to the operating system, waiting for the next time the counter counts down to zero.

Program 2 achieves this result by POKEing a machine language program into memory starting at the middle of page 6, location 1664 (\$680), and transferring control to it via the USR function. We use the upper half of page 6 because the lower half can be overwritten if an INPUT command receives 128 or more bytes at one time.

Analysis of the Program

Program 3 is the machine language version of the program that does all the work. After setting up the equates that identify the various registers in lines 20-40 and starting the assembly at location \$680 in line 50, we get down to setting ourselves up in business. Lines 80 to 170 pull the three parameters passed by the USR function off the stack and store them in the spaces we reserved for them in lines 260, 270, and 280. We will discuss these parameters further when we reach the points where they are used.

Lines 190 to 220 store the address of our routine that does the actual blinking in the two-byte vector CDRMA2 in the usual low-byte, high-byte order. Lines 230 and 240 take the value of the parameter we have called PERIOD and store it in the actual timer location CDTMV2. Since this is the value that is decremented each sixtieth of a second, it is clear that the parameter PERIOD is just what its name suggests: the period, in sixtieths of a second, of the blink. With this final act the USR function has completed its work, so it returns to BASIC with the RTS at line 250.

Lines 260 to 280 are the storage locations of the three parameters PERIOD, MASK, and FLIP; we already have seen the significance of PERIOD. The actual blink routine is simplicity itself. CHACT is loaded into the A register (line 300), the value is ANDed with the bits in MASK (line 310) to eliminate any bits that we do not want, and the remaining bits are exclusively ORed with the bits in FLIP (line 320) and restored in CHACT (line 330). We can now see the significance of the parameters MASK and FLIP: they define the bits of CHACT that we wish to use and toggle.

The routine ends by resetting the timer for the next period (lines 350, 360) and returning to the operating system vertical blank routines. After this, it is ready to wait for PERIOD more vertical blanks and then do it all over again.

The BASIC program that POKes the machine language into place does not have to remain in memory once it has done its work. It may be removed with a NEW statement, and a different program that uses the blinking characters can be loaded. In fact, the call to the USR function in line 30 of the BASIC program may be eliminated, and a different program may turn on the blinking. Pressing <SYSTEM RESET> will stop the blinking, but another call to USR (1664, PERIOD, MASK, FLIP) will restore it.

You may experiment with the effect of toggling the various bits of CHACT by using different values of MASK and FLIP. Changing MASK to 23 and leaving FLIP at 1 causes the inverse video to remain on during the blanking. If both MASK and FLIP are changed to 3, inverse video is on while the characters are displayed, but the blanks are normal blanks. Setting both parameters to 2 produces an alternation between regular and inverse video that is quite eye-catching. Finally, setting MASK and FLIP to 4 causes an effect that you will just have to see for yourself; I still haven't figured out what this is used for, but it is spectacular. Of course, PERIOD may be set to any value between 1 and 255 that you wish to vary the rate with which the characters change.

Since "second stage" vertical blank routines are suspended whenever IO is in progress, you will see that the blinking stops during any disk or cassette activity (or anything that uses the serial IO bus for that matter). You can achieve some unique effects with this short program, and I am sure that many novel programs will use this in ways that I have never thought of.

Program 1. Blinking Characters

```
10 CHACT=755
20 DELAY=200
30 PRINT "HELLO"
40 FOR I=1 TO DELAY:NEXT I
50 POKE CHACT,0
60 FOR I=1 TO DELAY:NEXT I
70 POKE CHACT,1
80 GOTO 40
90 END
```

Program 2. Character Blink Routine

```
10 FOR I=1664 TO 1718
```

```

20 READ B:POKE I,B:NEXT I
30 A=USR(1664,30,1,1)
40 END
50 DATA 104,104,104,141,161,6,104,104,141,16
  2,6,104,104,141,163,6,169,164,141,40,2,16
  9,6,141,41
60 DATA 2,173,161,6,141,26,2,96,0,0,0
70 DATA 173,243,2,45,162,6,77,163,6,141,243,
  2,173,161,6,141,26,2,96

```

Program 3. Machine Language Version

```

0010 ;CHARACTER BLINK ROUTINE
0020 CHACT      = $2F3
0030 CDTMV2     = $21A
0040 CDTMA2     = $228
0050           *= $0680
0060 ;PULL PARAMETERS FROM STACK
0070 ;AND STORE THEM
0080           PLA
0090           PLA
0100           PLA
0110           STA PERIOD
0120           PLA
0130           PLA
0140           STA MASK
0150           PLA
0160           PLA
0170           STA FLIP
0180 ;STORE VECTOR TO BLINK ROUTINE
0190           LDA #BLINK&$00FF
0200           STA CDTMA2
0210           LDA #BLINK/256
0220           STA CDTMA2+1
0230           LDA PERIOD
0240           STA CDTMV2
0250           RTS
0260 PERIOD     *=*+1
0270 MASK       *=*+1
0280 FLIP       *=*+1
0290 ;HERE IS THE BLINK ROUTINE
0300 BLINK      LDA CHACT
0310           AND MASK
0320           EOR FLIP
0330           STA CHACT
0340 ;RESET TIMER AND RETURN
0350           LDA PERIOD
0360           STA CDTMV2
0370           RTS

```

String Arrays

Stephen Levy

It is possible to simulate string arrays in Atari BASIC. The illustrations here show how.

"If you want string arrays on your Atari computer, you'll just have to purchase Atari's Microsoft BASIC disk." A common belief, but not entirely true. You *can* create a string array using Atari BASIC. Microsoft BASIC does make the handling of arrays much easier, but it is possible to create a string array in Atari BASIC.

Creating the Array

What you will actually be creating is a long string which will hold all the elements of the array. In order that the array not have garbage in it, we must clean it out before using it.

There are two ways to clean out the string. The program below simply DIMensions a string to 1000 and then fills the string with "*" using a FOR/NEXT loop. Then it prints the string.

```
100 DIM B$(1000)
110 FOR A=1 TO 1000
120 B$(A,A)="*"
130 NEXT A
140 PRINT B$
```

The next program does the same thing a little differently and much more efficiently.

```
100 DIM B$(1000)
110 B$="*":B$(1000)="*":B$(2)=B$
120 PRINT B$
```

A lot faster, isn't it? You can use this method anytime you want to fill a large string with the same character. That is exactly what we must do to begin creating our string array. But this time we need to fill the string with blanks.

Enter and RUN the program below. When the program asks for names, enter the names of ten friends, pressing RETURN after each. The program as written will allow names with up to ten letters.


```

100 DIM ARRAY$(100),ELEMENT$(10):PRINT CHR$(
125)
110 ARRAY$=" ":ARRAY$(100)=" ":ARRAY$(2)=ARR
AY$
120 FOR A=1 TO 10
130 PRINT "NAME FOR ARRAY$(";A;") PLEASE";:I
NPUT ELEMENT$
140 ARRAY$(A*10-9,A*10)=ELEMENT$
150 ELEMENT$=" ":NEXT A
160 PRINT
200 FOR A=1 TO 10
210 PRINT "ARRAY$(";A;") IS ";ARRAY$(A*10-9,
A*10):NEXT A
300 TRAP 340
310 PRINT :PRINT "GIVE THE NUMBER (1 TO 10)"
320 PRINT "OF THE ARRAY YOU WISH TO SEE";:IN
PUT A
330 PRINT ARRAY$(A*10-9,A*10):GOTO 310
340 PRINT CHR$(253):GOTO 300

```

Notice that the program sets up an array with ten elements and allows you to pick from any of the ten. Let's look more closely at how it is done.

Line 100 DIMensions the array and clears the screen. Line 110 fills the array with blanks. Line 120 tells the computer to do it ten times. Line 130 gets your input.

Line 140 is the heart of the creation of the array. Within the parentheses the computer is told what *part* of the string should hold your input string ELEMENT\$. The first time through $A = 1$; therefore, $ARRAY$(A*10 - 9, A*10)$ will mean $ARRAY$(1, 10)$, or the first 10 positions in the string. When $A = 2$, we place ELEMENT\$ in the positions 11 to 20 ($2*10 - 9 = 11$ and $2*10 = 20$). We will continue to do this until the string is full.

Line 210 does the same thing, but in reverse order: it reads ARRAY\$ and prints the proper part to the screen. Line 330 also does the same thing, but only for the part of the string you request.

Try this: RUN the program and enter any ten names. Then press BREAK. Type PRINT ARRAY\$ without a line number, press RETURN, and see what happens.

Now RUN the program again, but simply press RETURN without entering anything for the names. Notice that there appears to be nothing in ARRAY\$. That is not really true—it is filled with blanks. Type PRINT ARRAY\$ again and see what happens.

You might wonder what function lines 300 and 340 serve. Those two lines prevent the program from crashing when an incorrect INPUT is entered. TRAP 340 sends the program to line 340 instead of printing "ERROR 8 Line 320" when you enter a Q (or whatever) but the program requires a number between 1 and 10. PRINT CHR\$(253) rings the buzzer, just as PRINT CHR\$(125) in line 100 clears the screen.

Armed with this little bit of information, you now should be able to use string arrays in your own programs.

2 Sound

2

Sound Experimenter

Matt Giwer

If you've wanted more control over your Atari's sound, here's a solution. You can use this program to experiment, to add sound to other programs (via the SOUND or POKE instructions), and to govern all four voices and all aspects of special effects.

Sound is one of the most important capabilities of the Atari computer. Not only does it permit four-part harmony if you are so inclined, but sound is an essential ingredient in games. It transports you into the world of the game, filling your ears with the sound of a laser cannon, letting you hear force shields as they collapse around you.

Unfortunately, the sound commands are among the most difficult to experiment with. The SOUND instruction can sometimes be clumsy and inconvenient; for one thing, the sounds stay on until you turn them off with another SOUND instruction. Also, you can't achieve the full range of sound with the BASIC instruction, since using it changes any settings in AUDCTL (the register which controls sound effects).

Sound control is a complicated matter, and simple programs cannot offer you complete control over the sounds. Joysticks couldn't govern four channels with nine registers.

This program takes a little practice to get used to, but it permits total control over all sound registers plus AUDCTL, turns the channels on individually, and shuts them all off at once when you need silence. When you are satisfied with the sounds, you can display the appropriate BASIC statements in either the POKE or the SOUND format.

An Overview

Lets first briefly summarize the Atari sound system. (For complete details, see the *Atari Personal Computer System Hardware Manual*, pages III.12 through III.14.) There are four independent sound channels whose distortion, frequency, and volume can be independently controlled. These are addressed by the SOUND instruction with the numbers 0 through 3. The *Hardware Manual*

refers to them as 1 through 4. The sound data can be independently POKEd into registers 53760 through 53767. The odd numbers control volume and distortion, and the even numbers control the frequency. Register 53768 is AUDCTL, which controls all of the sound channels in one way or another. If you use the BASIC SOUND instruction, any changes you may have made to AUDCTL are reset—AUDCTL is set to zero. Thus you do not have full control of the sounds with the SOUND instruction.

This program attempts to give you easy control over all of these parameters. Compromises to reduce complexity have been made in favor of the notation and numbers used in the SOUND instruction. See the *BASIC Reference Manual* for further information.

The figure shows the display that you will see upon RUNning and entering the commands. The first eight lines, numbered B7 through B0, are the bits in the AUDCTL Register. To change bit seven to 1, type B7 and RETURN. To change it back to zero, type B7 and RETURN again. These are technical changes that give no indication of what the new sound will be like. Experimentation is best. Suffice it to say that using B1 through B4 turns on both of the sound channels associated with bit seven.

To discuss the next five lines of the figure, we have to jump down to the lines labeled D: and X:. There are two types of entries to make this program, those which are purely commands and those which require numbers. If you need to enter a number, enter the number first and press RETURN. If it is a pure command, simply enter the command and RETURN. If you wish to work with sound channel zero, type the following sequence: 0, RETURN, REG, RETURN. A 0 will appear after SOUND (REG)ISTER on the display. For a pure tone, type 10, RETURN, DIS, RETURN, and a 10 will appear after (DIS)TORTION:. Similarly, 100, RETURN, FRE, RETURN, and 8, RETURN, VOL, RETURN, will complete this part of the display.

To hear this sound, type 0, RETURN, CH, RETURN, and to turn it off, type OFF, RETURN. To see the POKE values for this sound, type PDIS, RETURN, and the list of nine POKes will appear on the screen. Copy these POKes into your program, and you will duplicate the sound that you hear. The top right POKE is AUDCTL. The next four rows are channels 0 through 3—the left column is the distortion and volume, and the right is the frequency for each channel.

If AUDCTL is 0—which is the same as bits B0 through B7

being all 0—then the SOUND instruction may be used. To see the SOUND instructions, type SDIS, RETURN, and the POKes will be replaced with SOUNDS.

The “force” output is in the odd-numbered POKE registers and produces a click from the TV. It is turned off and on by use of FRC, RETURN. If you have set any of the AUDCTL bits, you must use the POKes to duplicate the sounds. The sound channels must be turned on individually by the CH command. OFF turns off all channels. If you make a change and want to hear it, type the channel number and CH again. This may seem cumbersome, but otherwise the sounds would always be on.

Screen Display

```

AUDCTL (REG)ISTER 4
      9 BIT POLY: (B7):  0
clock Ch.0 w/1.79 MHz: (B6):  0
clock Ch.2 w/1.79 MHz: (B5):  0
      clock Ch.1 w/Ch.0: (B4):  0
      clock Ch.3 w/Ch.2: (B3):  0
clock Ch.0 w/Ch.2 HiP: (B2):  0
clock Ch.1 w/Ch.3 HiP: (B1):  0
      15 kHz: (B0):  0
      SOUND (REG)ISTER 0
      (DIS)TORTION:      10
      (FRE)QUENCY:      100
      FORCE OUTPUT:      0
      (VOL)UME:          8

X:
D:  ?■
REG DIS FRE FRC VOL
OFF CH
PDIS SDIS      POKE 53768, 0
POKE 53761, 168 POKE 53760, 100
POKE 53763, 0   POKE 53762, 0
POKE 53765, 0   POKE 53764, 0
POKE 53767, 0   POKE 53766, 0

```

Sound Experimenter

```

80 DIM S(5,8),IN$(50)
90 FOR I=0 TO 8:FOR J=0 TO 5:S(J,I)=0:NEXT J
  :NEXT I
100 REG=5000:DIS=5100:FRE=5200:FRC=5300:OFF=
    5400

```

```

102 CLD=5900:CLX=6000:VOL=6100:POKAUD=6200:C
   H=6300:START=6400:REGDIS=6500:BUZZ=6600
104 PDIS=6700:SDIS=6800:EDIS=6900
1000 REM DISPLAY
1002 GRAPHICS 0:POKE 752,1
1008 POSITION 2,0:? "AUDCTL (REG)ISTER 4"
1010 POSITION 2,1:? "{11 SPACES}9 BIT POLY:(B
   7):"
1020 POSITION 2,2:? "clock Ch.0 w/1.79 MHz:(
   B6):"
1030 POSITION 2,3:? "clock Ch.2 w/1.79 MHz:(
   B5):"
1040 POSITION 2,4:? "{4 SPACES}clock Ch.1 w/
   Ch.0:(B4):"
1050 POSITION 2,5:? "{4 SPACES}clock Ch.3 w/
   Ch.2:(B3):"
1060 POSITION 2,6:? "clock Ch.0 w/Ch.2 HiP:(
   B2):"
1070 POSITION 2,7:? "clock Ch.1 w/Ch.3 HiP:(
   B1):"
1080 POSITION 2,8:? "{15 SPACES}15 kHz:(B0):"
1090 POSITION 2,9:? "{5 SPACES}SOUND (REG)IS
   TER"
1100 POSITION 2,10:? "{6 SPACES}(DIS)TORTION
   :"
1110 POSITION 2,11:? "{7 SPACES}(FRE)QUENCY:
   "
1120 POSITION 2,12:? "{6 SPACES}FORCE OUTPUT
   :"
1126 POSITION 2,13:? "{10 SPACES}(VOL)UME:"
1128 POSITION 2,14:? "X:"
1130 POSITION 2,15:? "D:"
1140 POSITION 2,16:? "REG DIS FRE FRC VOL"
1150 POSITION 2,17:? "OFF CH"
1160 POSITION 2,18:? "PDIS SDIS"
1500 GOSUB START
2000 REM JUMP TABLE
2008 FOR ZZZ=1 TO 2 STEP 0
2010 POSITION 5,15:POKE 752,0:INPUT IN$:POKE
   752,1
2020 TRAP 2040:A=VAL(IN$):TRAP 40000
2030 POSITION 5,14:? A:GOSUB CLD
2040 IF IN$="REG" THEN GOSUB REG
2042 IF IN$="DIS" THEN GOSUB DIS
2044 IF IN$="FRE" THEN GOSUB FRE
2046 IF IN$="FRC" THEN GOSUB FRC
2048 IF IN$="OFF" THEN GOSUB OFF
2049 IF IN$="CH" THEN GOSUB CH
2058 IF IN$="VOL" THEN GOSUB VOL

```

```
2060 IF IN$="B7" THEN S(4,7)= NOT (S(4,7)):P
    OSITION 30,1:? S(4,7):GOSUB CLD
2061 IF IN$="B6" THEN S(4,6)= NOT (S(4,6)):P
    OSITION 30,2:? S(4,6):GOSUB CLD
2062 IF IN$="B5" THEN S(4,5)= NOT (S(4,5)):P
    OSITION 30,3:? S(4,5):GOSUB CLD
2063 IF IN$="B4" THEN S(4,4)= NOT (S(4,4)):P
    OSITION 30,4:? S(4,4):GOSUB CLD
2064 IF IN$="B3" THEN S(4,3)= NOT (S(4,3)):P
    OSITION 30,5:? S(4,3):GOSUB CLD
2065 IF IN$="B2" THEN S(4,2)= NOT (S(4,2)):P
    OSITION 30,6:? S(4,2):GOSUB CLD
2066 IF IN$="B1" THEN S(4,1)= NOT (S(4,1)):P
    OSITION 30,7:? S(4,1):GOSUB CLD
2067 IF IN$="B0" THEN S(4,0)= NOT (S(4,0)):P
    OSITION 30,8:? S(4,0):GOSUB CLD
2070 IF IN$="PDIS" THEN GOSUB PDIS
2072 IF IN$="SDIS" THEN GOSUB SDIS
2980 IF FAIL=1 THEN GOSUB BUZZ
2989 FAIL=0
2990 NEXT ZZZ
5000 REM REG REGISTER SET
5010 IF A<0 OR A>3 THEN FAIL=1
5020 IF A>0 OR A<4 THEN POSITION 24,9:? A
5030 C=A:REM S(C,B)
5040 GOSUB REGDIS
5088 GOSUB CLD:GOSUB CLX
5090 RETURN
5100 REM DIS DISTORTION LEVEL
5110 IF A<0 OR A>14 THEN FAIL=1:GOTO 5180
5112 IF INT(A/2)-A/2<>0 THEN FAIL=1:GOTO 518
    0
5120 IF A=0 THEN D1=0
5121 IF A=2 THEN D1=32
5122 IF A=4 THEN D1=64
5123 IF A=6 THEN D1=96
5124 IF A=8 THEN D1=128
5125 IF A=10 THEN D1=160
5126 IF A=12 THEN D1=192
5127 IF A=14 THEN D1=224
5130 POSITION 21,10:? A
5140 S(C,1)=D1:S(C,5)=A
5170 S(C,8)=A
5180 GOSUB CLD:GOSUB CLX
5190 RETURN
5200 REM FRE FREQUENCY STORE
5210 IF A<0 OR A>255 THEN FAIL=1
5218 POSITION 21,11:? "{8 SPACES}"
5220 POSITION 21,11:? A
```

```
5230 S(C,2)=A
5280 GOSUB CLD:GOSUB CLX
5290 RETURN
5300 REM FRC SET FORCE BIT
5310 IF A=0 THEN S(0,3)= NOT S(0,3)
5320 IF A=1 THEN S(1,3)= NOT S(1,3)
5330 IF A=2 THEN S(2,3)= NOT S(2,3)
5340 IF A=3 THEN S(3,3)= NOT S(3,3)
5350 POSITION 21,12: ? S(C,3)
5380 GOSUB CLD
5390 RETURN
5400 REM OFF TURN OFF SOUND
5410 POKE 53761,0:POKE 53763,0:POKE 53765,0:
POKE 53767,0
5480 GOSUB CLD
5490 RETURN
5900 REM CLD CLEAR D POS.
5910 POSITION 5,15: ? "{20 SPACES}"
5990 RETURN
6000 REM CLX CLEAR X POS.
6010 POSITION 5,14: ? "{21 SPACES}":A=0
6090 RETURN
6100 REM VOL VOLUME SET
6110 IF A<0 OR A>15 THEN FAIL=1:GOTO 6180
6120 POSITION 21,13: ? "{12 SPACES}"
6122 POSITION 21,13: ? A
6130 S(C,4)=A
6180 GOSUB CLD:GOSUB CLX
6190 RETURN
6200 REM POKAUD POKE AUDCTL VALUE
6208 SUM=0
6210 IF S(4,0)=1 THEN SUM=SUM+1
6211 IF S(4,1)=1 THEN SUM=SUM+2
6212 IF S(4,2)=1 THEN SUM=SUM+4
6213 IF S(4,3)=1 THEN SUM=SUM+8
6214 IF S(4,4)=1 THEN SUM=SUM+16
6215 IF S(4,5)=1 THEN SUM=SUM+32
6216 IF S(4,6)=1 THEN SUM=SUM+64
6217 IF S(4,7)=1 THEN SUM=SUM+128
6220 POKE 53768,SUM
6290 RETURN
6300 REM CH TURN ON SOUND CHANNELS
6310 GOSUB POKAUD
6320 IF A=0 THEN POKE 53761,S(0,1)+S(0,4):PO
KE 53760,S(0,2)
6322 IF A=1 THEN POKE 53763,S(1,1)+S(1,4):PO
KE 53762,S(1,2)
6324 IF A=2 THEN POKE 53765,S(2,1)+S(2,4):PO
KE 53764,S(2,2)
```

```
6326 IF A=3 THEN POKE 53767,S(3,1)+S(3,4):PO
      KE 53766,S(3,2)
6380 GOSUB CLX:GOSUB CLD:GOSUB REGDIS
6390 RETURN
6400 REM START SET UP
6410 FOR I=1 TO 8:POSITION 30,I:? "0":NEXT I
6490 RETURN
6500 REM REGDIS DISPLAY OF REGISTER
6505 POSITION 21,12:? "{3 SPACES}"
6506 POSITION 21,12:? S(C,3)
6510 POSITION 21,11:? "{6 SPACES}"
6511 POSITION 21,11:? S(C,2)
6520 POSITION 21,10:? "{6 SPACES}"
6521 POSITION 21,10
6522 IF S(C,1)=224 THEN ? "14"
6523 IF S(C,1)=192 THEN ? "12"
6524 IF S(C,1)=160 THEN ? "10"
6525 IF S(C,1)=128 THEN ? "8"
6526 IF S(C,1)=96 THEN ? "6"
6527 IF S(C,1)=64 THEN ? "4"
6528 IF S(C,1)=32 THEN ? "2"
6529 IF S(C,1)=0 THEN ? "0"
6530 POSITION 21,13:? "{6 SPACES}"
6531 POSITION 21,13:? S(C,4)
6590 RETURN
6600 REM BUZZ
6610 ? "{BELL}"
6690 RETURN
6700 REM PDIS DISPLAY OF POKE DATA
6705 GOSUB EDIS
6710 POSITION 20,18:? "POKE 53768, ";SUM
6720 POSITION 2,19:? "POKE 53761, ";S(0,1)+S
      (0,4):POSITION 20,19:? "POKE 53760, ";S
      (0,2)
6730 POSITION 2,20:? "POKE 53763, ";S(1,1)+S
      (1,4):POSITION 20,20:? "POKE 53762, ";S
      (1,2)
6740 POSITION 2,21:? "POKE 53765, ";S(2,1)+S
      (2,4):POSITION 20,21:? "POKE 53764, ";S
      (2,2)
6750 POSITION 2,22:? "POKE 53767, ";S(3,1)+S
      (3,4):POSITION 20,22:? "POKE 53766, ";S
      (3,2)
6780 GOSUB CLD
6790 RETURN
6800 REM SDIS DISPLAY OF SOUND DATA
6810 POSITION 2,19:? "SOUND 0, ";S(0,2);", "
      ;S(0,8);", " ;S(0,4)
```

```
6820 POSITION 2,20:? "SOUND 1, ";S(1,2);", "
      ;S(1,8);", ";S(1,4)
6830 POSITION 2,21:? "SOUND 2, ";S(2,2);", "
      ;S(2,8);", ";S(2,4)
6840 POSITION 2,22:? "SOUND 3, ";S(3,2);", "
      ;S(3,8);", ";S(3,4)
6880 GOSUB CLD
6890 RETURN
6900 REM EDIS  ERASE PDIS &SDIS
6910 POSITION 20,18:? "{18 SPACES}"
6920 POSITION 2,19:? "{35 SPACES}"
6930 POSITION 2,20:? "{35 SPACES}"
6940 POSITION 2,21:? "{35 SPACES}"
6950 POSITION 2,22:? "{35 SPACES}"
6990 RETURN
7000 END
```

16-Bit Music

Fred Tedsen

Did you know that you can improve the tuning of your Atari's notes and extend its range dramatically? Normally you can only choose among 256 notes with the ordinary SOUND command. These subroutines let you have more than 65,000 frequencies to make music that's more precise and more pleasant to hear.

As I listened to my Atari play a new song that I had entered from a magazine listing, I could hear that some of the notes were not quite right. The music extended into the third octave above middle C, and though the tune was recognizable, some of the notes were off pitch enough to make listening to the tune unpleasant. I decided that it was time for me to investigate 16-bit music. What I discovered was not only that the accuracy of the notes could be improved dramatically, but also that the effective range could be more than doubled.

How SOUND Works

Before we discuss 16-bit music, let's take a look at what is happening when we use the SOUND statement, or in other words, eight-bit sound, in Atari BASIC. The following registers in the POKEY chip are used for sound generation:

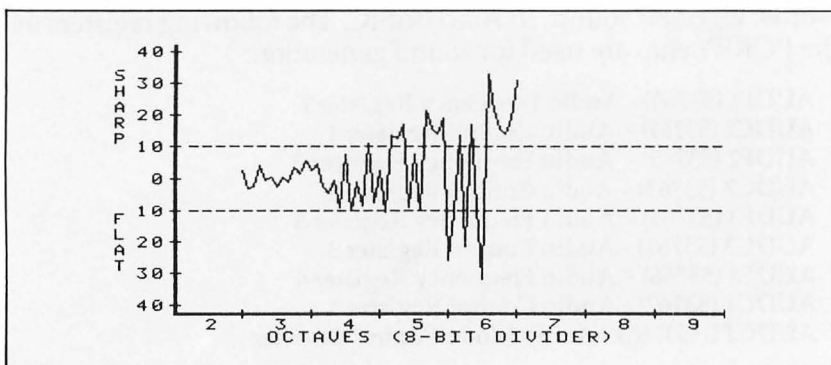
- AUDF1 (53760) - Audio Frequency Register 1
- AUDC1 (53761) - Audio Control Register 1
- AUDF2 (53762) - Audio Frequency Register 2
- AUDC2 (53763) - Audio Control Register 2
- AUDF3 (53764) - Audio Frequency Register 3
- AUDC3 (53765) - Audio Control Register 3
- AUDF4 (53766) - Audio Frequency Register 4
- AUDC4 (53767) - Audio Control Register 4
- AUDCTL (53768) - Audio Mode Control Register

The audio control registers are used to set volume (low order four bits) and sound content (high order bits). Thus there are 16 different volume settings and a variety of sounds available. For this discussion we are concerned only with pure tones, corresponding to SOUND x,x,10,x.

The audio frequency registers are used to control the divide by N circuits. These circuits use the contents of the frequency registers to divide a "clock" frequency to produce different output frequencies. Since they are one-byte registers, they are referred to as eight-bit dividers. The output frequency is determined by the formula $F_0 = F / (2 \times (\text{AUDF} + 1))$, where F is the clock frequency and AUDF the value in the audio frequency register. With a normal clock rate of 64 kilohertz (or more exactly 63,921 cycles per second), the frequency range is about 125 hertz to 32 kilohertz.

The effective range for music is limited to about four octaves. This is because the tuning accuracy of notes being reproduced becomes progressively worse as the frequency gets higher. Figure 1 illustrates this very clearly. It shows how far out of tune, measured in "cents," each note in the four octave range is. (A cent is 1/100 of a half-step. A sound which is 50 cents sharp or flat is exactly halfway between two notes.) Notes which are less than ten cents out of tune are usually acceptable, though two notes played together could sound bad if their combined inaccuracy is too large. For example, if you play a note which is eight cents flat followed by a higher note which is eight cents sharp, the second note will probably sound out of tune.

Figure 1. Tuning Inaccuracy of Musical Notes in Cents Using 8-Bit Dividers



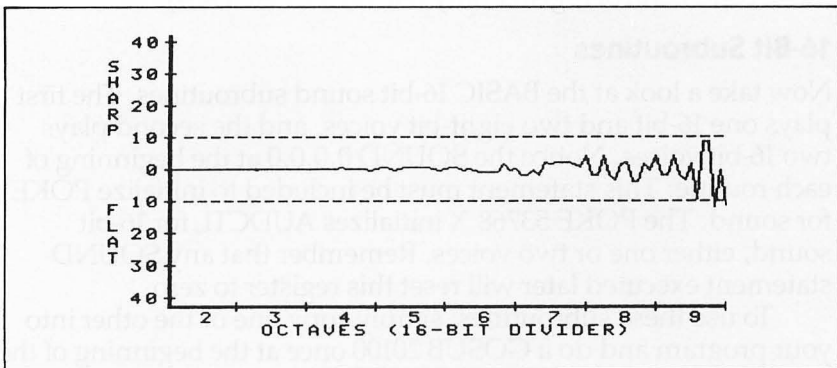
Tuning inaccuracy results from having a limited number of values to use as dividers. With an eight-bit divider, only 256 unique frequencies can be reproduced. The A note in the fourth octave should be 440 cycles per second. To reproduce this note on the

Atari, the number 72 is used as a divider. The resulting frequency is 437.8 hertz, which is 8.6 cents flat. If instead we use 71 as a divider, the output frequency is 443.9 hertz. This note is 15.3 cents sharp and is obviously a poorer choice than the note using 72. The choices become more restricted as the notes get higher. For the A note in the sixth octave, for example, 17 produces a note which is 15.3 cents sharp, while 18 produces a note 78.4 cents flat (closer to G# than A).

Fine-tuning: 16-Bit Dividers

Luckily, the Atari provides a solution to this problem: 16-bit dividers. With a 16-bit divider 65,536 different output frequencies are possible. For example, to reproduce the A in octave 6, we could use either 502 (1.8 cents flat) or 501 (1.6 cents sharp) and not be able to hear any difference. Figure 2 shows how dramatically the range and accuracy are improved.

Figure 2. Tuning Inaccuracy of Musical Notes in Cents Using 16-Bit Dividers



More accurate tuning does not come without a price. Sixteen-bit dividers are obtained by combining frequency registers: AUDF1 with AUDF2, or AUDF3 with AUDF4. This gives us a choice of one 16-bit and two eight-bit voices, or two 16-bit voices. We also cannot use the SOUND statement, even for the eight-bit voices, as it will confuse our settings for 16-bit sound. As it turns out, this is not much of a problem since machine language routines to play the music are simple and have the added advantage of being faster than separate SOUND statements.

Now let's look at how 16-bit sound is set up. The audio mode control register has four bits for this purpose:

Bit 6—Clock channel 1 with 1.79 megahertz instead of 64 kilohertz

Bit 5—Clock channel 3 with 1.79 megahertz

Bit 4—Combine channels 1 and 2

Bit 3—Combine channels 3 and 4

The other bits in AUDCTL have no bearing on this discussion, so we will ignore them. If you are curious, see Chapters 2 and 3 in the *Hardware Manual*.

The 1.79 megahertz (1.78979 megahertz, to be exact) clock rate is required to obtain the full range of output frequencies. The formula for determining output frequency is a little different: $F0 = F / (2 \times (AUDF + 7))$. In this case, AUDF is the two-byte frequency register value. The second register of the pair is the low order byte, either AUDF2 or AUDF4. For example, to use 1049 as a divider with registers 1 and 2, we would POKE 4 in AUDF2 and 25 in AUDF1.

The audio control register of the low order frequency register is not used and should be set to zero. Volume is controlled with the second control register only (AUDC2 or AUDC4).

16-Bit Subroutines

Now take a look at the BASIC 16-bit sound subroutines. The first plays one 16-bit and two eight-bit voices, and the second plays two 16-bit voices. Notice the SOUND 0,0,0,0 at the beginning of each routine. This statement must be included to initialize POKEY for sound. The POKE 53768,X initializes AUDCTL for 16-bit sound, either one or two voices. Remember that any SOUND statement executed later will reset this register to zero.

To use these subroutines, simply copy one or the other into your program and do a GOSUB 20100 once at the beginning of the program. Then, to play music, do the appropriate machine language call, $X = \text{USR}(\text{ADR}(\text{HF1\$}), N1, V1, N2, V2, N3, V3)$ or $X = \text{USR}(\text{ADR}(\text{HF2\$}), N1, V1, N2, V2)$. Nx is the note to be played and Vx is the volume. N1 is the 16-bit voice in the three-voice routine. You don't need to pass parameters for unused voices. For example, if you want only the 16-bit voice in the three-voice routine, you can use $X = \text{USR}(\text{ADR}(\text{HF1\$}), N1, V1)$, but to use only an eight-bit voice you would have to use $X = \text{USR}(\text{ADR}(\text{HF1\$}), 0, 0, N2, V2)$.

The note tables give you the most accurate values for four octaves of eight-bit and nine octaves of 16-bit notes. In a practical sense, the first octave of 16-bit notes is not usable because there

are some loud harmonics which tend to mask the actual note being played. You can get some good sounds if you hook up to a stereo amplifier, however. Notice that the eight-bit value for F# in the third octave is 172 rather than 173 as shown in the *BASIC Reference Manual*. 173 produces a note which is more than 12 cents flat, while the note from 172 is only 2.4 cents flat.

16-Bit and 8-Bit Note Table

NOTE	16-BIT	8-BIT		NOTE	16-BIT	8-BIT	
C	27357		OCTAVE 1	C	3414	121	OCTAVE 4
C#	25821			C#	3222	114	
D	24372			D	3040	108	
D#	23003			D#	2869	102	
E	21712			E	2708	96	
F	20493			F	2555	91	
F#	19342			F#	2412	85	
G	18256			G	2276	81	
G#	17231			G#	2148	76	
A	16264			A	2027	72	
A#	15351			A#	1913	68	
B	14489			B	1805	64	
C	13675		OCTAVE 2	C	1703	60	OCTAVE 5
C#	12907			C#	1607	57	
D	12182			D	1517	53	
D#	11498			D#	1431	50	
E	10852			E	1350	47	
F	10243			F	1274	45	
F#	9668			F#	1202	42	
G	9125			G	1134	40	
G#	8612			G#	1070	37	
A	8128			A	1010	35	
A#	7672			A#	953	33	
B	7241			B	899	31	
C	6834	243	OCTAVE 3	C	848	30	OCTAVE 6
C#	6450	230		C#	800	28	
D	6088	217		D	755	26	
D#	5746	204		D#	712	25	
E	5423	193		E	672	23	
F	5118	182		F	634	22	
F#	4830	172		F#	598	21	
G	4559	162		G	564	19	
G#	4303	153		G#	532	18	
A	4061	144		A	501	17	
A#	3832	136		A#	473	16	
B	3617	128		B	446	15	

NOTE	16-BIT	8-BIT		NOTE	16-BIT	8-BIT
C	421	14	OCTAVE 7	G	136	
C#	397			G#	128	
D	374			A	120	
D#	353			A#	113	
E	332			B	106	
F	313					
F#	295			C	100	OCTAVE 9
G	278			C#	94	
G#	262			D	88	
A	247			D#	83	
A#	233			E	78	
B	219			F	73	
				F#	69	
C	207		OCTAVE 8	G	64	
C#	195			G#	60	
D	183			A	57	
D#	173			A#	53	
E	163			B	50	
F	153					
F#	144			C	46	OCTAVE 10

Finally, some thoughts on when to use 16-bit music. If you have a piece of music which sounds fine using SOUND in BASIC, don't bother changing it—you probably won't be able to hear much improvement. I think you'll find that just about any music which extends into the fifth octave will be worth converting, however, especially if it is very complex. For three-part music, use the 16-bit voice for the highest notes. Some chord combinations may still sound slightly out of tune, in which case you might want to tune the 16-bit voice for the highest notes. Some chord combinations may still sound slightly out of tune, in which case you might want to tune the 16-bit voice a little sharp or flat to match the eight-bit voices. The large number of divider values available gives you plenty of possibilities.

Program 1. 16-Bit Sound Routine

```

200000 REM 16-BIT SOUND ROUTINE 1
200100 REM
200200 REM 1 16-BIT & 2 8-BIT VOICES
200300 REM
200400 REM X=USR(ADR(HF1$),N1,V1,N2,V2,N3,V3)
200500 REM
201000 SOUND 0,0,0,0:X=64+16:POKE 53768,X
201100 DIM HF1$(56):RESTORE 20140

```

```

20120 FOR I=1 TO 56:READ X:HF1$(I,I)=CHR$(X)
      :NEXT I
20130 RETURN
20140 DATA 104,170,104,141,2,210,104,141,0,2
      10,104,104,41,15,9,160,141,3,210
20150 DATA 224,2,240,32,104,104,141,4,210,10
      4,104,41,15,9,160,141,5,210
20160 DATA 224,4,240,14,104,104,141,6,210,10
      4,104,41,15,9,160,141,7,210,96

```

Program 2. 16-Bit Sound Routine 2

```

20000 REM 16-BIT SOUND ROUTINE 2
20010 REM
20020 REM 2 16-BIT VOICES
20030 REM
20040 REM X=USR(ADR(HF2$),N1,V1,N2,V2)
20050 REM
20100 SOUND 0,0,0,0:X=(64+16)+(32+8):POKE 53
      768,X
20110 DIM HF2$(41):RESTORE 20140
20120 FOR I=1 TO 41:READ X:HF2$(I,I)=CHR$(X)
      :NEXT I
20130 RETURN
20140 DATA 104,170,104,141,2,210,104,141,0,2
      10,104,104,41,15,9,160,141,3,210
20150 DATA 224,2,240,17
20160 DATA 104,141,6,210,104,141,4,210,104,1
      04,41,15,9,160,141,7,210,96

```


3 Applications and Education

3

Beginner's Keyboard

Marty Albers

Here is a short, simple program that gives very young computer users an entertaining introduction to the keyboard.

Software for young children is hard to find. Most kids' games and educational software are difficult for the preschooler to understand. Relating screen movement to joystick control can be a hard concept to grasp. I wrote the following program for my two-year-old so he would not feel left out when the rest of the family used the computer. It was designed to be easy to use (just push a key), educational (learn letters and numbers), and entertaining (colors and sound), and to provide a friendly start into the world of computer literacy.

Beginning programmers will also find this program rewarding, with some useful ideas to try on their own. An explanation of some of the less obvious lines follows. On lines 10 and 35 you will see one method of keyboard input without selecting the RETURN key (see "Reading the Atari Keyboard on the Fly," *COMPUTE!'s First Book of Atari*). Line 20 POKEs address 16 with 112 to disable the BREAK key. Line 45 allows larger characters in Graphics modes 1 and 2 by using the PRINT #6; statement. Also on line 45 is a conversion of the keyboard input from ATASCII code to character format: CHR\$(I).

Two sound registers are used (line 50), one with pure tones and one with distortions. Don't forget to turn the sounds off (line 51). The "default colors" are used, a black screen and four others for the characters. To find the other colors, use the Atari logo key and shift between the upper- and lowercase. The RETURN key is used in line 41 to start a new row of characters. When the screen is full, you start again in line 42. Now begin!

Beginner's Keyboard

```
1 REM : I=INPUT, L=LINE, R=ROW, T=TIME
10 OPEN #1,4,0,"K: "
15 GRAPHICS 2+16:L=0
20 POKE 16,112
25 FOR R=0 TO 20: IF R=20 THEN R=0:L=L+1
30 POSITION R,L
```

```
35 GET #1,I
40 IF I=155 AND I=11 THEN GOTO 15
41 IF I=155 THEN R=-1:L=L+1
42 IF R=18 AND L=11 THEN GOTO 15
45 PRINT #6;CHR$(I)
50 SOUND 0,2*I,10,8:SOUND 1,2.5*I,8,10:FOR T
   =1 TO 75:NEXT T
51 SOUND 0,0,0,0:SOUND 1,0,0,0
55 NEXT R
```

Spelling Quiz

Edward Perrin

Here is an educational program that will help students learn their weekly spelling words. Word lists can be SAVED to disk or tape for practice later. Requires 32K for disk and 16K for tape.

"Spelling Quiz" allows you or your child to enter weekly spelling words into the computer and save them on tape or disk. All the words for the year can be saved at once, or each week can be saved separately (20 words at a time) as the school year progresses.

The program allows you to enter up to 20 words at a time. I have found that most weekly spelling assignments are no more than 20 words, so this works out rather well. The program will accept fewer than 20 words, but no more than 20.

The program prompts are self-explanatory, but it would be good to read through the following instructions.

Load in the program with the BASIC cartridge inserted. The loading time for tape is about four minutes.

Type in RUN. After the title page you will be asked if you want to Create or Retrieve a list of words to work on. You will also be asked if you are using a Disk or Tape.

Creating Word Lists

To create a list, simply type in up to 20 words, no more than 20 letters each and with no leading or trailing spaces, one at a time, and hit the RETURN key. Be sure each word is spelled correctly before hitting RETURN. If you enter fewer than 20 words, type in an * following the last input. After the last word or the * you will be asked to type in some sort of identifier for that particular list. Use "Chapter 4" or "List 189," etc. Use an identifier that your child will understand. The identifier is used to make sure your child has retrieved the correct list.

Disk users will be asked to enter a filename. Only the filename is necessary; the program will supply the "D1:". Be sure to make the name unique and meaningful.

Tape users will need a blank tape or a tape which has been used to SAVE other word lists. Be sure to note the tape counter

number on a sheet of paper and to store the paper with the word tape.

If you already have words stored, just follow the prompts to LOAD the words.

Check the list and the identifier to be sure that this is the list you wanted to use. If not, you have the option to LOAD a new list or create a new one as needed.

Once the words are LOADED in with the Create or Retrieve option, your child is ready to use the program. You now choose one of three options: spell a Certain number of words correctly, spell an unlimited number of words correctly, or End.

If you choose the C option, you will be graded, and the program will terminate when the number of words spelled correctly equals the number you entered at the prompt. If you choose the unlimited option (by pressing RETURN), you can spell only 10,000 words before the program terminates. It is easy to change the 10,000 to another upper limit. Change the number in the last line of the program to stop the program automatically at a preset number.

The Quiz Begins

When you have made all of the necessary choices, the game is ready to play. The screen will show the number of the word being scrambled, the score (the number of words spelled correctly), a scrambled word, and the attempt number. At the bottom of the screen is a GRAPHICS 0 window where you will type your answers. The word number on top will help the child who cannot figure out the scrambled word. The program checks spelling competence rather than ability to unscramble words, so there is no penalty for not unscrambling correctly. Use this option as you wish.

Your child will then have three tries to spell the word correctly. If the spelling is correct, the screen will respond with an encouraging CORRECT and a happy sound. After three tries, the program will give the correct spelling and set up a different screen to allow the child to practice the misspelled word.

Practice Screen

The practice screen will not allow misspellings. It does allow the child to press the * to exit when he or she wants to. After each word in this mode, be sure to press the space bar, not RETURN.

Pressing RETURN will cause the computer to register an error in the spelling.

After the number of correct spellings equals the number put in at the beginning, or if your child enters * instead of spelling a word during the main run, the quiz ends and your child is graded on his performance. If you think the grading is too strict, change the limits in the grading subroutine in line 9000-9999.

After the grading, your child can go back and retrieve, or create and save a new file, or use the words already in the computer's memory. Your child has the option to end at this time. If he or she continues, the whole cycle repeats.

Spelling Quiz

```

2 REM SPELLING QUIZ
10 DIM A$(20),B$(20),C$(20),D$(20),E$(20),F$(20),G$(20),H$(20),I$(20),J$(20),K$(20),L$(20),M$(20),N$(20),O$(20)
20 DIM P$(20),Q$(20),R$(20),S$(20),T$(20),U$(20),ARRAY(20),Z$(20),STANDINGS(20),ANS$(1),WORD$(520),INWORD$(128)
30 DIM WEL$(38),DK$(15),ZZ$(1):TIME=0
40 GOSUB 3000
54 GOSUB 13000
55 PRINT "HOW MANY WORDS DO YOU WISH TO SPELL {3 SPACES} CORRECTLY BEFORE ENDING THIS D RILL? {3 SPACES} ENTER 0 TO END PROGRAM"
56 TRAP 56:INPUT RIGHT:IF RIGHT=0 THEN 4000
57 TRAP OFF:SCORE=0:ATT=0
58 GOSUB 5200:TRAP OFF
59 W1=0:W2=0:W3=0:W4=0:W5=0:W6=0:W7=0:W8=0:W9=0:W10=0:W11=0:W12=0:W13=0:W14=0:W15=0:W16=0:W17=0:W18=0:W19=0:W20=0
60 IF SCORE=RIGHT THEN 1000
65 NUM=1:W=INT(20*RND(1)+1)
70 GOSUB 7000:IF A$="{20 SPACES}" THEN NUM=-1:GOSUB 7000:GOTO 65
80 GRAPHICS 2:POKE 708,0:FOR AR=1 TO 20:ARRAY(AR)=-1:NEXT AR:PRINT "{BELL}";"INPUT '*' TO END QUIZ"
90 POSITION 11,3:?"#6;"SCORE";" ";SCORE:POSITION 2,0:?"#6;"#";W;" ON YOUR LIST"
110 FOR L=1 TO 20:IF A$(L,L)="" THEN L=L-1:A$=A$(1,L):GOTO 115
112 NEXT L:IF L=21 THEN L=20
115 FOR LTR=1 TO L
120 ARR=INT(L*RND(0)+1):IF ARRAY(ARR)=1 THEN
120

```

```

125 P=ARR-1
130 POSITION P,7:? #6;A$(LTR,LTR):ARRAY(ARR)
    =1:NEXT LTR:POKE 708,200
135 TRY=0
137 TRY=TRY+1:ATT=ATT+1:POSITION 0,9:? #6;"A
    TTEMPT # ";ATT
140 INPUT Z$:IF Z$="*" THEN ATT=ATT-1:NUM=-1
    :GOSUB 7000:GOTO 1000
141 IF Z$=A$ THEN SCORE=SCORE+1:FOR N=100 TO
    10 STEP -1:SOUND 0,N,10,10:NEXT N:SOUND
    0,0,0,0
142 IF Z$=A$ THEN POSITION 12,6:? #6;"correct
!!":FOR N=1 TO 300:NEXT N:GOTO 60
143 IF TRY=3 THEN FOR N=1 TO 100:SOUND 0,20,
    4,10:NEXT N:SOUND 0,0,0,0:GOTO 2000
144 POSITION 0,2:? #6;"wrong...":POSITION 0,
    3:? #6;"try again..."
145 FOR N=1 TO 100:SOUND 0,11,4,10:NEXT N:SO
    UND 0,0,0,0
147 POSITION 0,2:? #6;"{8 SPACES}":POSITION
    0,3:? #6;"{11 SPACES}"
150 GOTO 137
1000 GRAPHICS 18:PRINT #6;"  your score is "
    ;SCORE
1010 PRINT #6:PRINT #6;"time to quit for now
    "
1020 ? #6:? #6:? #6;"{3 SPACES}CONGRATULATIO
    NS"
1030 ? #6;"on a job well done!!":? #6:? #6;"
QUIZ WILL REPEAT!"
1040 GOSUB 5000:SOUND 0,0,0,0:SOUND 1,0,0,0:
    GOTO 9000
2000 POSITION 0,5:? #6;A$:POSITION 0,4:? #6;
    "answer:" :NUM=-1:GOSUB 7000
2010 FOR N=1 TO 1000
2020 NEXT N:GOSUB 8000
2030 GOTO 60
3000 GRAPHICS 18:POSITION 0,4:? #6;"
    {3 SPACES}SPELLING QUIZ"
3005 GOSUB 5000:SOUND 1,0,0,0:SOUND 0,0,0,0:
    RETURN
4000 GRAPHICS 18:POSITION 0,2:? #6;"VERY GOOD
    WORK..."
4010 POSITION 0,6:? #6;"see you again later"
4020 POSITION 4,10:? #6;"bye for now"
4030 GOSUB 5000:GOTO 7030
5000 FOR N=1 TO 200
5010 SOUND 0,RND(0)*200,10,2
5030 NEXT N

```

```

5040 RETURN
5100 FOR N=1 TO 100: SOUND 0,N,10,10: NEXT N: S
    OUND 0,0,0,0: RETURN
5200 FOR N=255 TO 200 STEP -1: SOUND 0,N,10,1
    0: NEXT N: FOR N=225 TO 150 STEP -1: SOUND
        0,N,10,10: NEXT N
5210 FOR N=175 TO 100 STEP -1: SOUND 0,N,10,1
    0: NEXT N: FOR N=150 TO 50 STEP -1: SOUND
        0,N,10,10: NEXT N: SOUND 0,0,0,0: RETURN
7000 IF W=1 THEN B$=WORD$(1,20): A$=B$: W1=W1+
    NUM
7001 IF W=2 THEN C$=WORD$(21,40): A$=C$: W2=W2
    +NUM
7002 IF W=3 THEN D$=WORD$(41,60): A$=D$: W3=W3
    +NUM
7003 IF W=4 THEN E$=WORD$(61,80): A$=E$: W4=W4
    +NUM
7004 IF W=5 THEN F$=WORD$(81,100): A$=F$: W5=W
    5+NUM
7005 IF W=6 THEN G$=WORD$(101,121): A$=G$: W6=
    W6+NUM
7006 IF W=7 THEN H$=WORD$(121,140): A$=H$: W7=
    W7+NUM
7007 IF W=8 THEN I$=WORD$(141,160): A$=I$: W8=
    W8+NUM
7008 IF W=9 THEN J$=WORD$(161,180): A$=J$: W9=
    W9+NUM
7009 IF W=10 THEN K$=WORD$(181,200): A$=K$: W1
    0=W10+NUM
7010 IF W=11 THEN L$=WORD$(201,220): A$=L$: W1
    1=W11+NUM
7011 IF W=12 THEN M$=WORD$(221,240): A$=M$: W1
    2=W12+NUM
7012 IF W=13 THEN N$=WORD$(241,260): A$=N$: W1
    3=W13+NUM
7013 IF W=14 THEN O$=WORD$(261,280): A$=O$: W1
    4=W14+NUM
7014 IF W=15 THEN P$=WORD$(281,300): A$=P$: W1
    5=W15+NUM
7015 IF W=16 THEN Q$=WORD$(301,320): A$=Q$: W1
    6=W16+NUM
7016 IF W=17 THEN R$=WORD$(321,340): A$=R$: W1
    7=W17+NUM
7017 IF W=18 THEN S$=WORD$(341,360): A$=S$: W1
    8=W18+NUM
7018 IF W=19 THEN T$=WORD$(361,380): A$=T$: W1
    9=W19+NUM
7019 IF W=20 THEN U$=WORD$(381,400): A$=U$: W2
    0=W20+NUM

```



```

7020 RETURN
7030 END
8000 GRAPHICS 18:POKE 708,100: ? #6;"please t
ype the word":POSITION 0,1: ? #6;A$
8005 POSITION 0,2: ? #6;"and hit spacebar..."
8007 POSITION 0,3: ? #6;"UNTIL SCREEN IS FULL
"
8010 POSITION 0,4: ? #6;"or type * to return"
:L=0:COUNTER=0
8100 OPEN #1,4,0,"K:"
8150 L=L+1
8200 GET #1,CHAR
8300 CLOSE #1
8350 IF CHR$(CHAR)="*" THEN GOTO 60
8355 IF CHR$(CHAR)=" " THEN L=0:GOSUB 8400
8360 IF CHR$(CHAR)<>A$(L,L) THEN GOSUB 12000
:GOTO 8000
8370 IF L=LEN(A$) THEN L=0
8400 COUNTER=COUNTER+1:PRINT #6;CHR$(CHAR);:
TRAP 40000
8450 IF COUNTER>139 THEN GOTO 60
8500 GOTO 8100
8600 RETURN
9000 POKE 752,1:PRINT "HERE IS A LIST OF HOW
MANY TIMES EACH WORD WAS SPELLED CORRE
CTLY THIS TIME."
9010 NUM=0:FOR W=1 TO 20:GOSUB 7000:NEXT W
9050 FOR N=100 TO 240:SOUND 0,N,10,10:NEXT N
:SOUND 0,0,0,0
9100 ? W1;" ";B$: ? W2;" ";C$: ? W3;" ";D$: ? W
4;" ";E$: ? W5;" ";F$: ? W6;" ";G$: ? W7;"
";H$: ? W8;" ";I$: ? W9;" ";J$: ? W10;" "
;K$
9150 ? W11;" ";L$: ? W12;" ";M$: ? W13;" ";N$:
? W14;" ";O$: ? W15;" ";P$: ? W16;" ";Q$:
? W17;" ";R$: ? W18;" ";S$: ? W19;" ";T$
9200 ? W20;" ";U$
9250 POKE 752,1:POSITION 25,3:PRINT "ATTEMPT
5":POSITION 28,5:PRINT ATT:FOR N=1 TO 2
00:SOUND 0,255,10,8:NEXT N
9260 POSITION 25,7:PRINT "CORRECT":POSITION
28,9:PRINT SCORE:FOR N=1 TO 200:SOUND 0
,200,10,8:NEXT N
9270 TRAP 9400:PER=INT((SCORE/ATT)*100):POSI
TION 25,11:PRINT "PERCENT":POSITION 28,
13:PRINT PER;"%"
9280 FOR N=1 TO 200:SOUND 0,100,10,6:NEXT N
9300 POSITION 25,15: ? "GRADE:"

```

```

9310 IF PER>=95 THEN POSITION 27,17: ? " A ":
    POSITION 25,21: ? " EXCELLENT!!! "
9320 IF PER>=88 AND PER<95 THEN POSITION 27,
    17: ? " B ": POSITION 25,21: ? " VERY GOOD!! "
9330 IF PER>=78 AND PER<88 THEN POSITION 27,
    17: ? " C ": POSITION 25,21: ? " GOOD!!! "
9340 IF PER>=70 AND PER<78 THEN POSITION 27,
    17: ? " D ": POSITION 25,21: ? " HMMMM!! "
9350 IF PER<70 THEN POSITION 27,17: ? " F ": P
    OSITION 25,21: ? " STUDY!!! "
9360 SOUND 0,0,0,0:POKE 752,0:GOTO 54
9400 PER=0:POSITION 28,13:PRINT PER:POSITION
    25,11:PRINT " PERCENT ":GOTO 9280
10000 WEL$=" WELCOME TO THE WORLD OF SPELLING
    QUIZ ":PRINT "{CLEAR}":FOR N=1 TO 37:P
    RINT WEL$(N,N);:NEXT N:TIME=TIME+1
10010 OFF=40000:P=0: ? : ? "DO YOU WANT TO CRE
    ATE OR RETRIEVE THE FILE";
10011 O$="{20 SPACES}":FOR N=1 TO 520 STEP 20
    :WORD$(N,N+19)=O$:NEXT N
10012 ? : ? : ? "ONCE YOU CREATE A FILE IT WIL
    L BE{5 SPACES}STORED ON TAPE OR DISK S
    O YOU CAN{5 SPACES}INPUT THE WORDS FRO
    M"
10013 ? "THE TAPE OR DISK INSTEAD OF TYPING
    {3 SPACES}THE SAME WORDS IN EVERY TIME
    YOU PLAY."
10014 ? : ? : ? "TYPE IN C OR R AND HIT RETURN
    NOW!"
10015 TRAP 10014:INPUT ANS$
10017 TRAP 10017: ? "ARE YOU USING TAPE OR DI
    SK":INPUT ZZ$:IF ZZ$(1,1)<>"T" AND ZZ$
    (1,1)<>"D" THEN 10017
10020 IF ANS$<>"C" THEN GOTO 11000
10100 ? "TYPE IN WORDS NOW":N=1
10105 ? : ? : ? " BE SURE EACH WORD IS SPELLED
    CORRECTLY BEFORE YOU PRESS RETURN
    {14 SPACES }"
10110 FOR N=1 TO 400 STEP 20:INPUT INWORD$
10120 IF N>399 THEN WORD$(401,520)=" ":GOTO
    10200
10125 IF INWORD$="*" THEN WORD$(N,520)=" ":G
    OTO 10150
10130 WORD$(N,N+19)=INWORD$
10140 NEXT N
10150 ? "TYPE IN CHAPTER # OR LIST # ETC..."
10160 INPUT INWORD$:WORD$(401,420)=INWORD$

```

```

10200 FOR N=1 TO 420 STEP 20:PRINT WORD$(N,N
+19):NEXT N
10202 IF ZZ$="D" THEN GOSUB 10500:TRAP 40000
:OPEN #2,8,0,DK$:GOTO 10209
10203 ? "POSITION THE TAPE AND TAKE NOTE OF
{4 SPACES}THE COUNTER NUMBER.":? :? "P
RESS THE PLAY AND RECORD BUTTONS."
10204 ? :? "WHEN THE BUZZER SOUNDS, PRESS RE
TURN"
10205 N=1
10206 TRAP 10207:LPRINT
10207 OPEN #2,8,0,"C:"
10209 N=1:FOR X=1 TO 4
10210 PRINT #2;WORD$(N,N+119):N=N+120
10220 NEXT X:CLOSE #2
10300 GOTO 13000
10500 PRINT "YOU MUST NOW ENTER THE FILENAME
{7 SPACES}(WITHOUT 'D:') OF THE FILE T
O ";
10520 IF ANS$="R" THEN PRINT "LOAD":GOTO 105
30
10525 PRINT "CREATE"
10530 TRAP 10500:INPUT DK$:DK$(4)=DK$:DK$(1,
3)="D1:"
10540 RETURN
11000 IF ZZ$="D" THEN GOSUB 10500:N=1:TRAP 4
0000:OPEN #2,4,0,DK$:GOTO 11025
11005 ? "TO LOAD WORDS THAT ARE STORED ON TA
PE BE SURE TO POSITION THE TAPE AT THE
{3 SPACES}CORRECT COUNTER # YOU NEED."
11010 ? "WHEN BUZZER SOUNDS, PRESS RETURN AN
D WAIT FOR THE WORDS TO BE LOADED INT
O THE COMPUTER..."
11020 N=1:OPEN #2,4,0,"C:"
11025 FOR X=1 TO 4
11030 TRAP 11040:INPUT #2,INWORD$
11035 WORD$(N,N+119)=INWORD$:N=N+120
11040 NEXT X
11045 CLOSE #2
11060 FOR N=1 TO 400 STEP 20:PRINT INT(N/20)
+1;" ";WORD$(N,N+19)
11067 NEXT N
11068 PRINT "{9 SPACES}";WORD$(401,420)
11070 ? "IS THIS THE GROUP OF WORDS THAT YOU
{3 SPACES}WANTED (Y/N)";:INPUT ANS$:IF
ANS$="N" THEN GOTO 10000
11075 GOTO 13019
11080 END

```

```

12000 FOR N=1 TO 100: SOUND 0,20,4,10:NEXT N:
    SOUND 0,0,0,0
12005 GRAPHICS 18:? #6;"You typed ";CHR$(CHA
R);" this is "
12010 POSITION 0,1:? #6;"wrong...TRY AGAIN"
12020 POSITION 4,3:? #6;"THE WORD IS "
12030 POSITION 0,4:? #6;A$
12040 POSITION 6,5:? #6;"READY???"
12050 L=0:FOR N=1 TO 400:NEXT N:RETURN
13000 WEL$="NOW YOU MUST MAKE A BIG DECISION
!!!":FOR N=1 TO 35:PRINT WEL$(N,N);:NE
XT N:NUM=1
13005 TRAP 13010:? :? "HIT THE RETURN KEY WH
EN READY";:INPUT A:IF A=0 THEN END
13010 PRINT "{CLEAR}{BELL}":TRAP OFF
13011 ? :? :? :? :? "DO YOU WANT TO USE THE
LIST OF WORDS ALREADY IN THE COMPUTER
OR DO YOU WANT TO LOAD IN A NEW LIST"
13012 ? :? :? "TYPE IN C FOR A NEW LIST OR H
IT RETURN TO USE THE OLD LIST.":? :? :
? "TYPE IN E TO END"
13013 ? :? :? "OF COURSE, IF THIS IS THE FIR
ST TIME THROUGH THE PROGRAM DURING TH
IS{7 SPACES}SESSION YOU MUST HIT C!!!"
13014 INPUT ANS$:IF ANS$="N" THEN GOTO 10000
13015 IF ANS$="E" THEN GOTO 4000
13016 IF TIME=0 THEN ? "{3 BELL}":GOTO 13018
13017 GOTO 13019
13018 ? :? :? "THIS IS YOUR FIRST TIME THROU
GH THE{3 SPACES}PROGRAM. YOU MUST LOAD
IN OR CREATE A NEW LIST NOW!":GOTO 13
011
13019 PRINT "{CLEAR}"
13020 ? :? :? "IF YOU WANT TO PRACTICE FOR A
CERTAIN NUMBER OF TIMES TYPE IN C AND
HIT{5 SPACES}RETURN."
13025 ? :? :? "IF YOU WANT TO PRACTICE UNTIL
YOU GET TIRED JUST HIT RETURN."
13030 ? :? :? "IF YOU WANT TO QUIT, TYPE IN
E."
13050 INPUT ANS$:IF ANS$="C" THEN GOTO 55
13060 IF ANS$="E" THEN GOTO 4000
13070 RIGHT=10000:GOTO 57

```

Elementary Numbers

Stephen Levy

This educational program for preschoolers requires children to use only the three console keys to answer questions.

When you bought your computer, one reason you used to justify the purchase was that the kids could use it for educational purposes. Well, now the computer is home, but the three-year-old rarely uses it. "Too young," you tell yourself, "maybe in a few years."

Children as young as two can and are using computers every day. But the lack of good software is still the major reason preschoolers don't make greater use of computers. To be used successfully with preschoolers, educational software must be truly educational, must have a difficulty level and subject matter appropriate for the age group, and must hold the child's attention and be fairly simple to use.

For the Very Young

Using computers to teach young children can be fun and challenging. The Atari's design makes it extremely easy for young children to use. Although the Atari offers numerous ways to input answers, this program, once LOADED, requires only the use of the three function keys to input responses. The subject matter, elementary numbers, is basic and is intended to teach the numbers from one to ten and the addition of single digits.

There are four options for the child to pick from. When the menu appears the youngster must use the function keys to select the part of the program to use. Pressing the SELECT key moves a small marker from one option to the next. When the child is satisfied with his or her selection, he or she presses the START key. It is important that the child hold down the key until the computer responds (this is true throughout the program).

The Options

The four options are Adding, Counting, Next Number, and Select a Number.

Adding presents a simple addition problem and an equivalent number of symbols for each number in the problem. By

5 ***** FIVE
+4 ##### FOUR

9 &&&&&&&&&&

In the final option, Select a Number, the child must match the word for a number with the correct number.

The computer is your tool; why not make it a learning tool for your children?

```

100 REM ELEMENTARY NUMBERS
110 REM
120 REM COMPUTE! PUBLICATIONS
130 DIM CLEAR$(1),NUMBER$(51),C$(1),NUM$(6)
140 NUMBER$="ZERO ONE TWO THREEFOUR FIVE S
    IX SEVENEIGHTNINE ":CLEAR$=CHR$(125):C$
    =CHR$(94)

```

```

150 GRAPHICS 18:SETCOLOR 4,8,3
160 POSITION 5,3:PRINT #6;"ELEMENTARY":POSIT
    ION 5,7:PRINT #6;" NUMBERS":GOSUB 360
170 AA=4
180 PRINT #6;CLEAR$:POSITION 0,0:PRINT #6;"P
    RESS start TO BEGIN":POSITION 0,1:PRINT
    #6;"PRESS select TO PICK"
190 POSITION 3,4:PRINT #6;"adding":POSITION
    3,6:PRINT #6;"counting":POSITION 3,8:PRI
    NT #6;"next number"
200 POSITION 3,10:PRINT #6;"select a number"
210 IF PEEK(53279)=5 THEN AA=AA+2
220 IF AA=12 THEN AA=4
230 IF AA=4 THEN POSITION 0,10:PRINT #6;" "
240 IF PEEK(53279)=6 THEN GOTO 260
250 POSITION 0,AA:PRINT #6;">>":POSITION 0,A
    A-2:PRINT #6;" ":GOSUB 360:GOTO 210
260 IF AA=4 THEN 500
270 IF AA=6 THEN 960
280 IF AA=8 THEN 1260
290 IF AA=10 THEN 1580
300 FOR AA=1 TO NUM1:POSITION AA+5,4:PRINT #
    6;C$:NEXT AA
310 POSITION AA+6,4:PRINT #6;NUMBER$(NUM1+1+
    (NUM1*4),NUM1+5+(NUM1*4)):RETURN
320 FOR AA=1 TO NUM2:POSITION AA+5,6:PRINT #
    6;C$:NEXT AA
330 POSITION AA+6,6:PRINT #6;NUMBER$(NUM2+1+
    (NUM2*4),NUM2+5+(NUM2*4)):RETURN
340 NUM1=INT(RND(0)*10):RETURN
350 NUM2=INT(RND(0)*10):RETURN
360 FOR WAIT=1 TO 500:NEXT WAIT:RETURN
370 IF AA=10 THEN 1370
380 GOTO 400
390 IF AA=19 THEN 520
400 IF AA<11 THEN POSITION 5+AA,8:PRINT #6;C
    $:POSITION 3,8:PRINT #6;AA
410 SOUND 0,75,10,8
420 IF AA=10 THEN POSITION 2,8:PRINT #6;"10
    "
430 IF AA>10 THEN POSITION 5+(AA-10),9:PRINT
    #6;C$:POSITION 2,8:PRINT #6;AA
440 SOUND 0,0,0,0
450 RETURN
460 NUM$=NUMBER$(COUNT+1+(COUNT*4),COUNT+5+(
    COUNT*4)):RETURN
470 CHAR=INT(RND(0)*8)+36:GOTO 490
480 CHAR=INT(RND(0)*5)+60
490 C$=CHR$(CHAR):RETURN

```

```

500 REM ADDING
510 GOSUB 340:GOSUB 350
520 GRAPHICS 18:SETCOLOR 4,14,12:SETCOLOR 0,
    8,18
530 POSITION 3,4:PRINT #6;NUM1
540 IF NUM1=0 THEN POSITION 5,4:PRINT #6;"ZG
    rc":GOTO 560
550 GOSUB 470:GOSUB 300
560 POSITION 3,6:PRINT #6;NUM2
570 IF NUM2=0 THEN POSITION 5,6:PRINT #6;"ZG
    rc":GOTO 590
580 GOSUB 480:GOSUB 320
590 POSITION 2,7:PRINT #6;"====":POSITION 1,5
    :PRINT #6;"+"
600 AA=0:POSITION 3,8:PRINT #6;"0"
610 GOSUB 470
620 POSITION 0,0:PRINT #6;"press select to
    {12 SPACES}change answer":GOSUB 360
630 IF PEEK(53279)=5 THEN AA=AA+1:GOSUB 390
640 POSITION 0,0:PRINT #6;"press option when
    you like your answer ":GOSUB 360
650 IF PEEK(53279)=5 THEN AA=AA+1:GOSUB 390
660 IF PEEK(53279)=3 THEN 680
670 GOTO 620
680 IF AA=NUM1+NUM2 THEN GOSUB 750
690 IF AA<>NUM1+NUM2 THEN GOSUB 770:GOTO 520
700 POSITION 0,0:PRINT #6;"press SELECT for
    {4 SPACES}another problem{8 SPACES}":GO
    SUB 360:GOSUB 360
710 IF PEEK(53279)=5 THEN 500
720 IF PEEK(53279)=6 THEN 150
730 POSITION 0,0:PRINT #6;"press start for m
    enu{17 SPACES}":GOSUB 360
740 GOSUB 360:GOTO 700
750 REM CORRECT ANSWER
760 POSITION 2,11:PRINT #6;"correct":GOSUB 1
    920:RETURN
770 REM WRONG ANSWER
780 POSITION 2,11:PRINT #6;"sorry, try again
    "
790 FOR S=1 TO 2
800 SOUND 0,120,2,8
810 GOSUB 950
820 SOUND 0,29,10,12
830 FOR WAIT=1 TO 40:NEXT WAIT
840 GOSUB 940:NEXT S
850 FOR S=1 TO 3
860 SOUND 0,180,2,8
870 GOSUB 950:GOSUB 940

```



```

880 NEXT S
890 FOR S1=1 TO 2
900 SOUND 0,29,10,11
910 FOR WAIT=1 TO 40:NEXT WAIT
920 GOSUB 940:NEXT S1
930 RETURN
940 SOUND 0,0,0,0:FOR WAIT=1 TO 40:NEXT WAIT
:RETURN
950 FOR WAIT=1 TO 80:NEXT WAIT:RETURN
960 REM COUNTING
970 TIMES=1
980 GOSUB 2130
990 SETCOLOR 4,8,5:SETCOLOR 0,9,14
1000 POKE 87,2:POSITION 5,2:PRINT #6;"COUNTI
NG"
1010 FOR C=1 TO 15:SETCOLOR 4,C,8:FOR WAIT=1
TO 25:SOUND 0,C*15,10,8:NEXT WAIT
1020 SOUND 0,0,0,0:NEXT C
1030 SETCOLOR 4,8,5:SETCOLOR 0,9,14:SETCOLOR
1,12,10
1040 POKE 87,2:PRINT #6;CLEAR$:Q=1
1050 COLOR 2:POKE 87,5:FOR C1=6 TO 8:PLOT 0,
C1:DRAWTO 79,C1:NEXT C1
1060 FOR COUNT=1 TO 9
1070 REM
1080 GOSUB 460
1090 POKE 87,1:POSITION 0,3:PRINT #6;NUM$:PO
SITION 15,3:PRINT #6;NUM$:POSITION 9,3:
PRINT #6;COUNT
1100 POKE 87,2:FOR C1=5 TO 13 STEP 4:POSITIO
N C1,0:PRINT #6;COUNT:NEXT C1
1110 POKE 87,2:POSITION 2,2:PRINT #6;NUM$:PO
SITION 9,1:PRINT #6;COUNT:POSITION 13,2
:PRINT #6;NUM$
1120 SETCOLOR 2,3,7
1130 COLOR 3
1140 SOUND 0,120,10,8
1150 POKE 87,5:PLOT Q+4,15:DRAWTO Q+4,11:DRA
WTO Q,11:POSITION Q,15:POKE 765,3:XIO 1
8,#6,0,0,"S:"
1160 SOUND 0,0,0,0
1170 Q=Q+8
1180 GOSUB 360
1190 COLOR 3
1200 POKE 87,5:PLOT Q+4,15:DRAWTO Q+4,11:DRA
WTO Q,11:POSITION Q,15:POKE 765,3:XIO 1
8,#6,0,0,"S:"
1210 NEXT COUNT
1220 TIMES=TIMES+1

```

```

1230 IF TIMES=3 THEN 150
1240 GOTO 980
1250 END
1260 REM NEXT NUMBER
1270 GOSUB 2130
1280 SETCOLOR 4,5,10:SETCOLOR 0,6,3:SETCOLOR
    1,11,6:SETCOLOR 2,3,3
1290 POKE 87,1:POSITION 5,4:PRINT #6;"NEXT N
    UMBER"
1300 FOR C1=2 TO 3
1310 COLOR C1:C2=C1*4
1320 POKE 87,5:PLOT 79,C2+2:DRAWTO 79,C2:DRA
    WTO 0,C2:POSITION 0,C2+2:POKE 765,C1:X
    I 0 18,#6,0,0,"S:"
1330 NEXT C1
1340 GOSUB 340:IF NUM1=9 OR NUM1=0 THEN 1340
1350 GOSUB 360:GOSUB 360
1360 GOSUB 480
1370 GRAPHICS 17
1380 POSITION 2,13:PRINT #6;"PRESS THE selec
    t":POSITION 2,15:PRINT #6;"KEY UNTIL YO
    U"
1390 POSITION 2,17:PRINT #6;"FIND THE":POSIT
    ION 2,19:PRINT #6;"NEXT NUMBER"
1400 POSITION 3,4:PRINT #6;NUM1:GOSUB 300
1410 AA=0
1420 IF PEEK(53279)=5 THEN AA=AA+1:GOSUB 370
1430 IF PEEK(53279)=3 THEN 1500
1440 POSITION 0,0:PRINT #6;"press option whe
    n you like your answer ":GOSUB 360
1450 GOSUB 470
1460 IF PEEK(53279)=5 THEN AA=AA+1:GOSUB 370
1470 POSITION 0,0:PRINT #6;"press select to
    {12 SPACES}change answer":GOSUB 360
1480 IF PEEK(53279)=3 THEN 1500
1490 GOTO 1420
1500 IF AA=NUM1+1 THEN GOSUB 750
1510 IF AA<>NUM1+1 THEN GOSUB 770:GOTO 1370
1520 IF AA<>NUM1+1 THEN 1370
1530 POSITION 0,0:PRINT #6;"press SELECT for
    {4 SPACES}another problem{8 SPACES}":G
    OSUB 360:GOSUB 360
1540 IF PEEK(53279)=5 THEN 1340
1550 IF PEEK(53279)=6 THEN 150
1560 POSITION 0,0:PRINT #6;"press start for
    menu{17 SPACES}":GOSUB 360
1570 GOSUB 360:GOTO 1530
1580 REM SELECT A NUMBER
1590 COUNT=INT(RND(0)*9):GOSUB 460

```

```

1600 GRAPHICS 18:SETCOLOR 4,5,9:SETCOLOR 0,7
    ,5
1610 POSITION 1,0:PRINT #6;"MATCH UP THE wor
    d":POSITION 2,1:PRINT #6;"WITH THE numb
    er"
1620 POSITION 0,8:PRINT #6;"PRESS start TO B
    EGIN"
1630 AA=1
1640 GOSUB 360
1650 IF PEEK(53279)<>6 THEN 1650
1660 GRAPHICS 18:SETCOLOR 0,1,13:SETCOLOR 4,
    5,9
1670 POSITION 8,7:PRINT #6;NUM$
1680 POSITION 2,3:PRINT #6;"select TO CHANGE
    ":POSITION 1,4:PRINT #6;"option IF YOU
    LIKE"
1690 POSITION 4,5:PRINT #6;"YOUR ANSWER"
1700 GOSUB 360
1710 POSITION 1,10:PRINT #6;"0 1 2 3 4 5 6 7
    8 9"
1720 IF PEEK(53279)=5 THEN AA=AA+2:SOUND 0,7
    5,10,8:FOR W=1 TO 10:NEXT W:SOUND 0,0,0
    ,0
1730 IF PEEK(53279)=3 THEN 1800
1740 IF AA>19 THEN AA=1:POSITION 19,9:PRINT
    #6;" "
1750 IF AA=1 THEN 1770
1760 POSITION AA-2,9:PRINT #6;" "
1770 POSITION AA,9:PRINT #6;C$
1780 GOSUB 360
1790 GOTO 1720
1800 ANS=((AA+1)/2)-1
1810 IF ANS=COUNT THEN GOSUB 750
1820 IF ANS<>COUNT THEN GOSUB 770:GOTO 1660
1830 GOSUB 360
1840 GRAPHICS 18:SETCOLOR 4,8,12:SETCOLOR 0,
    8,2
1850 POSITION 1,3:PRINT #6;"VERY GOOD":POSIT
    ION 2,5:PRINT #6;NUM$;" IS ";COUNT
1860 GOSUB 360:GOSUB 360
1870 POSITION 2,5:PRINT #6;"OPTION FOR MENU"
1880 POSITION 1,3:PRINT #6;"SELECT FOR ANOTH
    ER{6 SPACES} PROBLEM{7 SPACES}"
1890 IF PEEK(53279)=3 THEN 150
1900 IF PEEK(53279)=5 THEN 1590
1910 GOTO 1890
1920 REM INTRO MUSIC
1930 S3=2
1940 MUSIC=INT(RND(0)*2)+1

```

```
1950 RESTORE 5300+(MUSIC*100)
1960 READ S1,TIME
1970 IF S1=-1 THEN SETCOLOR 4,8,3:RETURN
1980 SOUND 0,S1+3,10,7:SOUND 1,S1,10,11
1990 SETCOLOR 4,S3,8
2000 FOR WAIT=1 TO TIME*7:NEXT WAIT
2010 SOUND 0,0,0,0:SOUND 1,0,0,0:FOR WAIT=1
    TO 3:NEXT WAIT
2020 S3=S3+2:IF S3>15 THEN S3=1
2030 GOTO 1960
2130 REM ROUTINE FOR MODE 2 3 ROWS
2140 REM MODE 1 2 ROWS
2150 REM MODE 5 32 ROWS
2160 GRAPHICS 5
2170 BEGIN=PEEK(560)+PEEK(561)*256+4
2180 POKE BEGIN-1,71
2190 POKE BEGIN+2,7:POKE BEGIN+3,7
2200 POKE BEGIN+4,6:POKE BEGIN+5,6
2210 POKE BEGIN+38,65:POKE BEGIN+39,PEEK(560)
    ):POKE BEGIN+40,PEEK(561)
2220 RETURN
5400 DATA 122,2,122,2,82,2,82,2,73,2,73,2,82
    ,4,92,2
5410 DATA 92,2,97,2,97,2,109,2,109,2,122,4
5420 DATA 82,2,82,2,92,2,92,2,97,2,97,2,109,
    4
5430 DATA 82,2,82,2,92,2,92,2,97,2,97,2,109,
    4
5440 DATA 122,2,122,2,82,2,82,2,73,2,73,2,82
    ,4
5450 DATA 92,2,92,2,97,2,97,2,109,2,109,2,12
    2,4,-1,-1
5500 DATA 122,2,109,2,97,2,122,2,122,2,109,2
    ,97,2,122,2,97,2,92,2,82,4,97,2,92,2,82
    ,4
5510 DATA 82,1,73,1,82,1,92,1,97,2,122,2,82,
    1,73,1,82,1,92,1
5520 DATA 97,2,122,2,122,2,82,2,122,4,122,2,
    82,2,122,4,-1,-1
```

Standings

Dan and Philip Seyer

"Standings" is a program for sports fans who would like to create their own standings statistics. It was written by a 12-year-old and his father.

This program will enable you to create professional-looking team standings statistics. We developed the program with baseball in mind, but you can use it for any sport. You might even adapt it for other purposes. (For example, a manager or supervisor might use it to keep track of employee performance.)

Input in Graphics Mode 2 + 16

Once you type in the program and get it working, you'll see a colorful menu in Graphics mode 2. After you type A, you will be prompted to enter the date, the name of the sport, the number of teams, the team names, and win-loss records. A special routine at lines 420 to 499 allows you to enter this data in Graphics mode 2. Normally, you can't enter data in this mode without using a text window and an INPUT statement.

Output Data

After you enter the data mentioned above, the program does the rest. It calculates each team's percentage and GB statistic. (GB stands for *games behind the leader*.) Then the program sorts the teams into proper order according to winning percentages.

If you hold down the OPTION key when you select choice B, the program will play some random sounds as it prints the sport caption at the top of the screen. If you get tired of hearing the sounds, just press B without holding down the OPTION key. Then the program will skip over the random sound-generation routine.

Updating Statistics

Statistics are easily updated. To do this, select option B from the main menu. You can then change a team's win-loss record by pressing W to add a win or L to add a loss. The team's percentage changes instantly when you change the wins or losses. You can also change the spelling of a team's name, delete an entire team

record, or add a new team. The program prompts you step by step for the appropriate entries and then modifies or deletes the appropriate DATA statements. The program will automatically resort the teams into proper order after you have updated all the win-loss statistics.

Resaving the Program

After updating the statistics, be sure to end the program by selecting the END option from the main menu. The program will then ask you whether you have a program recorder or disk drive. You can answer by typing P for program recorder or D for disk drive.

Program recorder. If you have a program recorder, you will be asked to position the tape for saving the program. When you press RETURN, your program will be saved.

Disk drive. If you have a disk drive, the program and any new data will automatically be resaved to disk when you type D and press RETURN.

Printout

To get a printout of your favorite league's standings, just type C for your menu choice. You will then be prompted to turn on your printer. (You may also want to adjust your paper at this time.) Then press RETURN to start the printing.

STATS ENTERED 08-28-83

HOMETOWN LEAGUE STANDINGS

	TEAMS	W	L	PCT.	GB
1	TIGERS	17	0	1.000	--
2	BEARS	9	8	.529	8
3	PADRES	9	9	.500	8 1/2
4	A'S	9	10	.474	9
5	RAMS	8	11	.421	10
6	LIONS	7	10	.412	10
7	SENATORS	6	11	.353	11
8	WHITE SOX	4	15	.211	14

Sort Routine

The teams will be listed in order by percentage from highest to lowest. A sort routine at line 900 to 972 does this for you automatically. Also notice the GB statistics, games behind the leader. The

GB statistic is the number of times, a team must beat the first place team to move into a tie for first place.

Self-Modifying Code

An interesting feature of the program is that it is self-modifying. When you enter information, the program creates DATA statements for you and saves the information in those DATA statements. (See lines 400 to 420.) This way, you don't need a separate data file since the data is saved along with your program.

The program as printed here contains the data for the eight teams listed in the sample printout. The sample data are included only to get you started. It is suggested that you practice with this data and experiment with the program. Then delete each of the eight teams and enter your own information.

Standings

```

0 ? "INITIALIZING....."
1 READ Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10:SAV=Q1
2 READ Q12,Q13,Q14,Q15,Q17,Q18,Q20,Q21,Q22,Q
  24,Q26,Q27,Q30,Q33,Q34,Q35,Q40,Q64,Q65,Q68
  ,Q70
3 READ Q72,Q74,Q82,Q89,Q95,Q97,Q100,Q125,Q12
  6,Q128,Q155,Q165,Q190,Q246,Q255,Q260,Q261,
  Q286,Q289,Q300,Q304,Q306
4 READ Q400,Q425,Q430,Q500,Q507,Q511,Q533,Q5
  59,Q578,Q630,Q694,Q705,Q752,Q760,Q764,Q765
  ,Q770,Q800,Q842,Q871,Q895
5 READ Q898,Q900,Q975:GOTO 450
6 DATA 1,2,3,4,5,6,7,8,9,10,12,13,14,15,17,1
  8,20,21,22,24,26,27,30,33,34,35,40,64,65,6
  8,70
7 DATA 72,74,82,89,95,97,100,125,126,128,155
  ,165,190,246,255,260,261,286,289,300,304,3
  06
8 DATA 400,425,430,500,507,511,533,559,578,6
  30,694,705,752,760,764,765,770,800,842,871
  ,895,898,900,975
13 RESTORE Q300:READ L,TEMP$:RETURN :REM GET
  S SPORT NAME
14 TEMP1$="{14 SPACES}":RETURN
15 GOSUB Q22:POSITION X,Y:IF Y$="%" THEN ? "
  █":GOTO Q17:REM INPUTS LETTER TYPED
16 ? #6;"█"
17 POSITION X,Y:GOSUB Q21:GET #Q1,A:RETURN
18 TEMP$="{15 SPACES}":RETURN
19 ? "{38 SPACES}";:RETURN

```

```

21 POKE Q764,Q255:RETURN :REM CLEAR5 KEY PRESSED REGISTER
22 CLOSE #Q1:OPEN #Q1,Q4,Q18,"K:":POKE Q752,Q1:RETURN :REM OPEN KEYBOARD FOR INPUT
24 II=INT(RND(Q0)*Q24):FOR A=Q12 TO Q0 STEP -Q1:SOUND Q0,II,Q2,A:NEXT A:RETURN :REM MAKES RANDOM SOUND
25 IF START=Q0 THEN RETURN :REM DELETES CHAR. FROM TEMPS
26 POSITION X,Y:IF Y$="%" THEN ? " "
27 IF Y$<>"%" THEN ? #6;" "
28 X=X-1:TRAP 31:TEMP$=TEMP$(Q1,START-Q1):START=START-Q1:RETURN
29 GOSUB Q22:GET #1,A:RETURN
31 START=Q0:X=XX:GOSUB Q18:RETURN
32 LL=ORDER(QL):RETURN
33 FOR D=YY+Q6 TO YY+Q13:POSITION Q1,D:GOSUB 19:NEXT D:POKE Q752,Q1:RETURN :REM CLEAR5 STAT MENU
35 POKE 559,0:FOR LNO=START TO LL STEP STEP: ? CHR$(125):REM LINE DELETION
36 RESTORE LNO:READ Y$:IF Y$="{ESC}" THEN RETURN
37 ? "{DOWN}";LNO: ? : ? : ? "CONT":POSITION Q0,Q0:POKE Q842,Q13:STOP
38 POKE Q842,Q12: ? CHR$(Q125):NEXT LNO:POKE 559,34:RETURN
39 GOSUB Q289:GOTO Q533
41 TEMP1$="Y":GOSUB Q705:GOSUB 32:REM TEAM DELETION
42 RESTORE LL:READ LNO,TEMP$:GOSUB Q33:POSITION Q6,YY+Q8
43 ? "TYPE Y TO DELETE ";TEMP$:POSITION Q6,YY+Q10: ? "HIT RETURN TO GO BACK TO MENU":GOSUB Q22:GET #Q1,A
44 IF A<>89 THEN RETURN
45 GOSUB Q260:TEAMNO=TEAMNO-Q1:LNO=Q304:TEMP1$=STR$(TEAMNO):GOSUB Q898:GOSUB Q400
46 START=LL:STEP=Q2:GOSUB Q35:IF TEAMNO=Q0 THEN GOTO Q507
47 GOSUB Q260:GOSUB Q975:GOSUB Q900:GOTO Q507
52 GRAPHICS Q18:GOSUB Q190:GOSUB Q24:POSITION Q3,Q3: ? #6;"turn on printer ":REM OUTPUT ROUTINE
53 ? #Q6: ? #Q6: ? #Q6;"{3 SPACES}HIT Y TO PRINT":GOSUB Q22:GOSUB Q97: ? #Q6
54 ? #Q6;"HIT RETURN FOR MENU":GOSUB Q22:GOTO 755

```



```

55 GOSUB Q97:RESTORE Q895:READ LNO,TEMP$:TRA
P 750:LPRINT :LINE$(Q10,Q24)="STATS ENTER
ED":X=LEN(TEMP$)
56 LINE$(Q26,LEN(TEMP$)+Q26)=TEMP$:LPRINT LI
NE$:GOSUB Q97:RESTORE 300:READ L,TEMP$:X=
(40-(LEN(TEMP$)+10))/2
57 LINE$(X,X+LEN(TEMP$))=TEMP$:LINE$(X+LEN(T
EMP$),40)=" STANDINGS"
58 LPRINT :LPRINT :LPRINT :LPRINT LINE$:REST
ORE Q871:READ L,L
59 RESTORE L:READ L,TEMP$,W,L:LPRINT :LPRINT
 "{3 SPACES}TEAMS{11 SPACES}W{3 SPACES}L
 {3 SPACES}PCT. GB":Y=Q0:GOSUB Q260:LPRIN
T
60 GOSUB Q97:FOR I=Q1 TO TEAMNO:Y$=")":RESTO
RE ORDER(I):READ LNO,TEMP1$,LW,LL:LINE$(Q
1,LEN(STR$(I)))=STR$(I)
61 LINE$(Q4,Q18)=TEMP1$:LINE$(Q20,Q20+LEN(ST
R$(LW)))=STR$(LW):LINE$(Q24,Q24+(LEN(STR$
(LL))))=STR$(LL)
62 GOSUB Q261:LINE$(Q27,Q33)=TEMP$(Q1,Q5):LI
NE$(34,Q40)=TEMP1$:LPRINT LINE$:GOSUB Q97
:NEXT I:RETURN
70 GOSUB Q260:IF TEAMNO>=Q30 THEN GOTO Q760:
REM ADDS TEAM
71 GRAPHICS Q18:GOSUB Q190:POSITION Q1,Q2:?
#6;"enter name of team":Y$="":GOSUB Q578
:LNO=Q306
72 RESTORE LNO:READ TEMP$:IF TEMP$<>STR$(LNO
) THEN GOSUB Q898:GOSUB Q400:ORDER(TeamNO
+Q1)=LNO:GOTO 74
73 LNO=LNO+Q2:GOTO Q72
74 TEAMNO=TEAMNO+Q1:TEMP1$=STR$(TEAMNO):LNO=
Q304:GOSUB Q400:GOSUB Q900:GOTO Q507
95 FOR I=1 TO 500:NEXT I:RETURN
97 LINE$="{38 SPACES}":RETURN
99 FOR D=Q6 TO Q22:POSITION Q1,D:GOSUB 19:NE
XT D:RETURN
100 GRAPHICS Q18:GOSUB Q190:GOSUB Q22:POSITI
ON Q0,Q4:? #Q6;"YOU MUST ENTER TEAMS BEFO
RE USING B OR C."
105 GOSUB Q95:GOTO Q507:REM GIVES MESSAGE
155 POKE Q82,Q2:POSITION Q2,YY+Q7:IF I<TEAMN
O THEN ? "E DISPLAY NEXT GROUP OF TEAMS
{7 SPACES}":REM STAT MENU
157 ? "E ADD OR SUBTRACT WINS OR LOSSES
{4 SPACES}":? "E RETURN TO MENU
{20 SPACES}:"
160 ? "E CORRECTION OF TEAM NAME{11 SPACES}":
? "E END{31 SPACES}:"

```

```

161 ? "F DELETE TEAM(23 SPACES)"
165 POKE Q764,Q255:GOSUB Q22:GET #Q1,A:IF A>
    Q64 OR A<Q72 THEN CLOSE #Q1:RETURN
170 GOTO Q165
190 POSITION Q5,Q0: ? #6;"STANDINGS":GOSUB Q2
    4:RETURN
199 REM DRAWS TITLE BOX
200 ? :POKE Q752,Q1:GOSUB Q13:L=LEN(TEMP$):L
    L=Q40-(STNO+L):LL=LL/Q2:POSITION LL,Q1:F
    OR I=Q1 TO L+STNO
205 ? "{N}";:NEXT I: ? "{DOWN}{LEFT}{B}":POSI
    TION LL,Q2: ? "{V}":POSITION LL,Q3:FOR I=
    Q1 TO L+STNO: ? "{M}";:NEXT I:RETURN
210 ? "■";:FOR J=Q1 TO LEN(TEMP$): ? TEMP$(J,
    J);:GOSUB Q24:NEXT J:RETURN
215 POSITION LL+Q1,Q2:FOR J=Q1 TO LEN(TEMP$)
    : ? TEMP$(J,J);:GOSUB Q24:NEXT J:RETURN
255 TEMP1$=TEMP$(Q1,START):RETURN
260 RESTORE Q304:READ LNO,TEAMNO:RETURN :REM
    PUTS NUMBER OF TEAMS INTO TEAMNO
261 IF LW=Q0 AND LL=Q0 THEN PCT1=Q0:GOTO 263
    :REM CALCULATES GB & PCT & PRINTS THEM
262 GOSUB Q14:GOSUB Q18:PCT1=(LW/(LW+LL)):PC
    T1=(PCT1+5E-04)
263 TEMP$(Q1,Q5)=STR$(PCT1):IF LW=Q0 THEN TE
    MP$=".000"
264 IF TEMP$(Q1,Q1)="0" THEN TEMP$(Q1,Q1)="
    "
265 IF TEMP$(Q3,Q3)=" " THEN TEMP$(Q3,Q3)="0
    "
266 IF TEMP$(Q4,Q4)=" " THEN TEMP$(Q4,Q4)="0
    "
267 IF TEMP$(Q5,Q5)=" " THEN TEMP$(Q5,Q5)="0
    "
268 IF TEMP$(Q2,Q2)=" " THEN TEMP$(Q2,Q2)="."
    "
269 GB=((W-L)/Q2)-(LW-LL)/Q2:IF L=Q0 THEN
    GB=W/Q2-(LW-LL)/Q2
270 IF GB<=Q0 THEN TEMP1$="--":GOTO Q286
271 TEMP1$=STR$(GB):FOR J=Q1 TO LEN(STR$(GB))
    :IF TEMP1$(J,J)<>"." THEN NEXT J:GOTO Q
    286
272 TEMP1$(J,J)=" ":TEMP1$(Q3,Q6)=" 1/2"
273 IF TEMP1$(Q1,Q1)="0" THEN TEMP1$(Q1,Q1)="
    "
286 IF Y$=")" THEN RETURN
287 POSITION Q26,Y+Q5: ? TEMP$(Q1,Q5):IF Y$<>
    "<" THEN POSITION Q33,Y+Q5: ? TEMP1$
288 RETURN

```

```

289 POSITION Q3,Q8: ? #6;" {3 SPACES}":RETURN
290 GOSUB Q289:POSITION Q0,Q10: ? #6;"
   {3 SPACES}too many teams{4 SPACES}":GOTO
   Q533
292 POSITION Q0,Y: ? #6;"NUMBERS ONLY, PLEASE
E":IF Y=Q8 THEN GOSUB Q770:GOTO 605
293 GOSUB Q765:GOTO 610
298 RESTORE ORDER(OL):READ LNO,TEMP1$,LW,LL:
   RETURN
299 REM NAME OF SPORT
300 DATA 300,HOMETOWN LEAGUE,0
302 DATA STANDINGS
303 REM NO. OF TEAMS
304 DATA 304,8,4,312
305 REM TEAM DATA
306 DATA 306,LIONS{9 SPACES},7,10
308 DATA 308,A'S{11 SPACES},9,10
310 DATA 310,BEARS{9 SPACES},9,8
314 DATA 314,RAMS{10 SPACES},8,11
316 DATA 316,TIGERS,17,0
318 DATA 318,PADRES{8 SPACES},9,9
320 DATA 320,WHITE SOX{5 SPACES},4,15
322 DATA 322,SENATORS,6,11
399 DATA {ESC},0,0
400 GRAPHICS Q0:POKE Q559,Q0:SETCOLOR Q1,Q9,
   Q4: ? CHR$(Q125):REM CREATES DATA LINES
410 ? "{DOWN}";LNO;" DATA ";LNO;" ";TEMP1$;"
   ";LW;" ";LL: ? : ? : ? "CONT"
415 POSITION Q0,Q0:POKE Q842,Q13:STOP
420 POKE Q842,Q12:SETCOLOR Q1,Q9,Q10:Y$="":R
   ETURN
425 W=Q0:START=Q0:XX=X:GOSUB Q18:POKE Q752,Q
   1:REM INPUT IN GR. MODE 2+16
430 GOSUB Q15:IF A=Q155 THEN GOTO 446
435 IF A=126 THEN GOSUB 25:GOTO Q430
440 IF START=PROP THEN GOTO Q430
441 IF W=Q1 THEN GOTO 445
442 IF LINE$="IN" AND A=Q27 THEN W=Q1:POKE Q
   694,Q128:GOTO Q430
443 IF LINE$<>"LINE" THEN POKE Q694,Q0
444 IF Y$="%" THEN ? CHR$(A):X=X+Q1:POSITION
   X,Y:START=START+Q1:TEMP$(START,START+Q1
   )=CHR$(A):GOTO Q430
445 ? #Q6;CHR$(A):X=X+Q1:POSITION X,Y:START=
   START+Q1:TEMP$(START,START+Q1)=CHR$(A):G
   OTO Q430
446 IF Y$<>"%" THEN POSITION X,Y: ? #6;" ":GO
   TO 448
447 POSITION X,Y: ? " "

```

```

448 IF START=Q0 THEN GOSUB Q18
449 POKE Q694,Q0:RETURN
450 DIM TEMP1$(Q14),TEMP$(Q15),Y$(Q1),ORDER(
Q30),LINE$(Q40),V$(1)
455 RESTORE Q800:READ Y$:IF Y$<>"{ESC}" THEN
  GOSUB Q260:RESTORE Q800:FOR I=Q1 TO TEA
MNO:READ W,W,L,L:ORDER(I)=W:NEXT I
500 REM MAIN MENU
507 V$="":XX=Q0:GRAPHICS Q18:GOSUB Q190:? #Q
6:? #Q6;"[E] enter new teams":GOSUB Q24:?
#Q6;"[E] look at standings"
508 START=1:GOSUB Q24:? #Q6;"[E] printout stan
dings";:GOSUB Q24
509 ? #Q6;"[C] add team":GOSUB Q24:? #Q6;"[E] en
d":GOSUB Q24:? #Q6
510 GOSUB Q260:? #Q6;" You have entered ";TE
AMNO:POSITION Q1,Q9:? #6;" teams."
511 POKE Q764,Q255:GOSUB Q22:GET #Q1,A:IF A<
Q64 OR A>Q70 THEN GOTO Q511
514 IF A=69 THEN GOSUB 1000:GOTO Q507
515 IF A=Q65 THEN GOTO 526
516 IF A=Q68 THEN GOTO Q70
517 RESTORE Q306:READ TEMP$:IF TEMP$(Q1,Q1)=
"{ESC}" THEN GOTO Q100
519 IF A=66 THEN SAV=Q1:GOSUB Q630:GOTO Q507
520 IF A=67 THEN GOSUB 52:GOTO Q507
522 GOTO Q511
523 REM ENTER TEAMS
526 GOSUB 875:LL=Q0:LW=Q0:GRAPHICS Q18:GOSUB
Q190:POSITION Q1,Q3:? #Q6;"enter name o
f sport"
527 LINE$="LINE":POKE Q694,Q128:X=Q3:Y=Q4:PR
OP=Q14:GOSUB Q425:TRAP 527:TEMP1$=TEMP$(
Q1,START)
530 GOSUB Q255:POSITION Q1,Q7:? #Q6;"enter n
o. of teams"
533 X=Q3:Y=Q8:PROP=Q2:GOSUB Q425:TRAP 39:TEA
MNO=VAL(TEMP$):IF TEAMNO>Q30 OR TEAMNO<1
THEN GOSUB 289:GOTO 533
535 LNO=Q300:GOSUB Q400:TEMP1$=TEMP$(Q1,STAR
T)
536 LNO=Q304:GOSUB Q400:START=Q306:LL=398:ST
EP=Q2:GOSUB Q35:GOSUB Q14
575 LNO=Q304:FOR I=Q1 TO TEAMNO:LNO=LNO+Q2:G
RAPHICS Q18:POKE 559,Q34:GOSUB 190:POSIT
ION Q1,Q2
576 ? #Q6;"enter team no. #";I
578 LINE$="IN":Y=Q3:X=Q3:PROP=Q14:GOSUB Q425
580 TEMP1$=TEMP$:IF START=Q0 THEN GOTO Q578

```

```

605 POSITION Q0,Q5:? #Q6;" enter no. of wins
":GOSUB Q24
606 X=Q3:Y=Q6:POSITION X,Y:PROP=Q3:GOSUB Q42
5:Y=Q8:TRAP Q770:LW=VAL(TEMP%):IF LW<Q0
OR LW>999 THEN GOTO Q770
610 POSITION Q0,Q8:? #Q6;" ENTER NO. OF LOSS
ES":GOSUB Q24
613 X=Q3:Y=Q9:POSITION X,Y:GOSUB Q425:Y=Q10:
TRAP Q765:LL=VAL(TEMP%):IF LL<Q0 OR LL>9
99 THEN GOTO Q765
614 IF Y$="" THEN RETURN
615 POKE 702,Q64:GOSUB Q400:NEXT I:GOSUB Q89
8:GOSUB Q975:GOSUB Q900:GOTO Q507
629 REM PRINTS STANDINGS IN GR. MODE G
630 Y$="":PCT1=Q0:RESTORE Q895:READ LNO,TEMP
$:POKE 82,(40-(LEN(TEMP%)+14))/2:GOSUB 7
90:? "STATS ENTERED";TEMP%
631 STNO=Q12:IF PEEK(53279)<>Q3 OR V$="Y" TH
EN GOSUB 200:POSITION LL+Q1,Q2:? TEMP%;"
STANDINGS":PCT1=1
632 IF V$="Y" THEN Y$="<"
635 IF PCT1=1 THEN GOSUB Q24:GOTO 650
640 GOSUB 200:GOSUB 215:RESTORE 302:READ TEM
P%:GOSUB 210
650 RESTORE Q871:READ L,L:RESTORE L:READ LNO
,TEMP%,W,L:POSITION Q0,Q5:? "{4 SPACES}T
EAMS{10 SPACES}{3 SPACES}{3 SPACES}PCT.
GB{5 SPACES}"
654 IF V$<>"Y" THEN START=1
655 Y=Q0:GOSUB Q260:FOR I=START TO TEAMNO:RE
STORE ORDER(I):READ LNO
659 Y=Y+Q1:POSITION Q1,Y+Q5:? I
660 POSITION Q4,Y+Q5:READ TEMP%:? TEMP%:READ
LW:POSITION 19,Y+Q5:? LW:READ LL
661 POSITION 23,Y+Q5:? LL:GOSUB Q261
676 IF I>24 THEN TEMP2=Q4:GOTO 680
677 IF I>16 THEN TEMP2=Q3:GOTO 680
678 IF I<Q9 THEN TEMP2=Q1:GOTO 680
679 TEMP2=Q2
680 IF I<>Q8 AND I<>16 AND I<>Q24 THEN NEXT
I
681 YY=Y:IF V$="Y" THEN 705
682 GOSUB Q155
695 GOSUB 1000:IF A=66 THEN GOTO Q705
696 IF A=Q65 AND I<TEAMNO THEN GOSUB 99:Y=Q0
:NEXT I
697 IF A=67 THEN RETURN
698 IF A=Q70 THEN GOTO 41
699 IF A=Q68 THEN 727

```

```

700 GOSUB Q165:GOTO 695
705 GOSUB Q33:Y$=" ":POSITION Q6,YY+Q7:? "EN
TER TEAM NO. THEN HIT RETURN":PROP=Q2:X
=Q17:Y=YY+Q8:POKE Q752,Q1
706 TRAP Q705:GOSUB Q425:GOSUB Q33:OL=VAL(TE
MP$):IF OL>TEAMNO THEN GOTO Q705
707 IF TEMP2=Q3 THEN IF OL<Q17 OR OL>Q24 THE
N GOTO Q705
708 IF TEMP2=Q2 THEN IF OL<Q9 OR OL>16 THEN
GOTO Q705
709 IF TEMP2=Q1 THEN IF OL>Q8 OR OL<Q1 THEN
GOTO Q705
710 IF TEMP2=4 THEN IF OL>30 OR OL<25 THEN G
OTO Q705
711 IF TEMP1$="Y" THEN RETURN
712 POKE Q82,Q2:POSITION Q2,YY+Q7:? "PRESS W
TO ADD WIN":? "PRESS L TO ADD LOSS":? "
PRESS S";:REM WIN LOSS CHANGE
713 ? " THEN W TO SUBTRACT WIN":? "PRESS S T
HEN L TO SUBTRACT LOSS":? "PRESS W FOR M
ENU":? "PRESS RETURN TO ";
714 ? "CHANGE ANOTHER TEAMS{8 SPACES}STATS":
GOSUB 298:STEP=OL:FOR I=1 TO YY:POSITION
33,I+5:? "{6 SPACES}";:NEXT I
715 IF OL>Q8 THEN OL=OL-Q8:GOTO 715
716 GOSUB 29:IF A<>Q155 AND A<>83 AND A<>77
AND A<>76 AND A<>87 THEN 716
717 GOTO 985:GOSUB 29:GOTO 720
718 IF A=87 THEN LW=LW-Q1:GOTO 723
719 IF A=76 THEN LL=LL-Q1:GOTO 723
720 IF A=76 THEN LL=LL+Q1
721 IF A=87 THEN LW=LW+Q1
722 IF A=77 THEN GOSUB Q33:POSITION Q8,D-Q2:
GOSUB 19:GOSUB 997:OL=STEP:GOSUB Q400:GO
SUB Q898:GOSUB Q900:RETURN
723 IF LL<Q0 THEN LL=Q0
724 IF LW<Q0 THEN LW=Q0
725 POSITION 19,OL+Q5:? LW;" ":POSITION 23,Q
L+Q5:? LL;" ":GOSUB Q21:Y=OL:Y$="<":GOSU
B Q261:GOTO 716
726 REM CHANGES TEAM NAME
727 TEMP1$="Y":GOSUB Q705:POSITION Q6,YY+Q7:
? "ENTER NEW NAME FOR TEAM #":OL:X=Q13:Y
=YY+Q8:PROP=Q14
728 LINE$="IN":Y$=" ":GOSUB Q425:TEMP1$=TEMP
$:RESTORE ORDER(OL):READ TEMP$,TEMP$,LW,
LL:LNO=ORDER(OL):GOSUB Q400
729 RETURN
730 IF A<>Q155 THEN Y=OL:GOSUB 272:GOTO 715

```

```

735 GOSUB Q33:POSITION Q0,YY+Q6:GOTO 681
750 TRAP 750:POSITION Q0,Q3:? #Q6;" turn on
  printer!":? #Q6:? #Q6;"{4 SPACES}then h
  it RE":POSITION Q0,Q6
755 ? #6;"{18 SPACES}":GOSUB Q22:GET #Q1,A:IF
  A<>80 THEN RETURN
756 GOTO 55
760 POSITION Q0,Q8:? #6;"sorry, you have
  {4 SPACES}already entered the maximum no
  . of teams":GOTO Q511
765 POSITION Q3,Q9:? #6;"{3 SPACES}":GOTO 61
  3
767 GOTO 56
770 POSITION Q3,Q6:? #6;"{3 SPACES}":GOTO 60
  6
775 ? :? " SEE ERROR - ";PEEK(195):POKE Q75
  2,Q0:END
790 GRAPHICS Q0:SETCOLOR Q2,Q12,Q4:SETCOLOR
  Q4,Q3,Q6:RETURN
799 REM DATA FOR THE ORDER OF TEAMS
800 DATA 800,316,4,860
802 DATA 802,310,4,860
804 DATA 804,318,4,860
806 DATA 806,308,4,860
808 DATA 808,314,4,860
810 DATA 810,306,4,860
812 DATA 812,322,4,860
814 DATA 814,320,4,860
871 DATA 871,316,4,860
875 GRAPHICS Q18:GOSUB Q190:POSITION Q0,Q2:?
  #Q6;"{4 SPACES}enter date":? #6;" EXAMP
  LE : 12/15/83"
880 ? #Q6;"{14 SPACES}DE":? #6;"{11 SPACES}DEC
  . 15"
885 LINE$="LINE":POKE Q694,Q128
890 X=Q6:Y=Q6:PROP=Q8:GOSUB Q425:LNO=Q895:TR
  AP 885:GOSUB Q255:GOSUB Q400:RETURN
895 DATA 895,06-30-83,9,10
898 GRAPHICS Q18:? #Q6:? #Q6:? #Q6;" sortin
  g teams.":? #Q6
899 ? #6;"{5 SPACES}PLEASE WAIT.":GOSUB Q95:
  RETURN
900 RESTORE ORDER(Q1):READ L,TEMP$,START,PRO
  P:TEAM=ORDER(Q1):START=START-PROP:GOSUB
  Q260:REM SORT ROUTINE
901 IF TEAMNO=Q0 THEN RETURN
902 FOR I=Q1 TO TEAMNO-Q1:REM SORT ROUTINE
903 FOR J=I+Q1 TO TEAMNO
910 RESTORE ORDER(I):READ LNO,TEMP$,W,L

```

```

915 IF W=Q0 THEN PCT1=Q0:GOTO 925
920 PCT1=(W/(W+L))
925 RESTORE ORDER(J):READ LNO,TEMP$,LW,LL
927 IF LW=Q0 THEN PCT2=Q0:GOTO 940
930 PCT2=(LW/(LW+LL))
940 IF LW-LL>START THEN TEAM=ORDER(J):START=
    LW-LL
945 IF PCT2=PCT1 AND W<LW THEN GOSUB 980:GOT
    O 971
950 IF PCT2>PCT1 THEN GOSUB 980
971 NEXT J:NEXT I
972 STEP=Q2:START=Q800:LL=860:GOSUB Q35:GOSU
    B Q260:LNO=798:REM STORES ORDER DATA
973 FOR I=Q1 TO TEAMNO:LNO=LNO+Q2:TEMP1$=STR
    $(ORDER(I)):GOSUB Q400:NEXT I:GOSUB Q260
    :TEMP1$=STR$(TEAM)
974 LNO=871:GOSUB 400:RETURN
975 POKE Q559,Q0:RESTORE Q306:FOR I=Q1 TO TE
    AMNO
976 READ LNO,TEMP1$,LW,LL:ORDER(I)=LNO:NEXT
    I
977 RETURN
980 PROP=ORDER(I):ORDER(I)=ORDER(J):ORDER(J)
    =PROP:RETURN
985 IF A=Q155 THEN 2000
990 IF A<>83 THEN 720
991 IF A=83 THEN POKE 764,255:GET #1,A:GOTO
    718
995 GET #1,A:GOTO 717
997 POSITION Q6,YY+Q7:? "ENTER DATE THEN HIT
    RETURN":PROP=Q8:Y$="%":X=16:Y=YY+Q8:POK
    E Q752,Q1:LINE$="LINE"
998 POKE Q694,Q128:RESTORE ORDER(OL):READ LN
    O,TEMP1$:GOSUB Q425:LNO=ORDER(OL):GOSUB
    Q400:GOSUB Q255:LNO=Q895
999 RETURN
1000 POKE 82,2:IF A<>69 THEN RETURN
1005 GOSUB 790:? "DO YOU HAVE A DISK DRIVE O
    R A PROGRAM RECORDER?:? :? "
    {8 SPACES}(HIT RETURN FOR MENU)"
1010 POSITION 16,5:INPUT TEMP$:TEMP Q507
1015 IF TEMP$(Q1,Q1)="D" OR TEMP$(Q1,Q1)="C"
    THEN 1050
1020 ? :? "{7 SPACES}HIT RETURN TO SAVE"
1025 CSAVE:END
1050 ? :? "{9 SPACES}HIT RETURN TO SAVE":GOS
    UB Q22
1055 GET #Q1,A:IF A<>155 THEN 1055
1060 TRAP 775:SAVE "D:STANDING.SAV":POKE 752
    ,0:END

```



```

2000 OL=STEP:RESTORE ORDER(OL):READ LNO,TEMP
    1$:LNO=ORDER(OL):GOSUB Q400:V$="Y"
2004 REM PICK TEAMS TO CHANGE STATS ON
2005 POKE 752,2:GRAPHICS 0:? :? :? " PICK G
ROUP OF TEAMS TO DISPLAY":? :?
2006 ? "{8 SPACES}1) TEAMS(1-8)"
2007 ? "{8 SPACES}2) TEAMS(9-16)"
2008 ? "{8 SPACES}3) TEAMS(17-24)"
2009 ? "{8 SPACES}4) TEAMS(25-32)":? :? :?
2010 TRAP 2010:? "{UP}{5 SPACES}YOUR CHOICE:
    {4 SPACES}{3 LEFT}";:INPUT CHOICE:IF CH
    OICE<1 OR CHOICE>4 THEN 2010
2015 GOSUB CHOICE+3000:IF START>TEAMNO THEN
    2010
2016 V$="Y":GOTO 630
3001 START=1:RETURN
3002 START=9:RETURN
3003 START=17:RETURN
3004 START=25:RETURN

```

CalCalc: Computerize Your Diet

Charles Brannon

This program can help you lose weight by cutting calories. Be sure to consult your doctor before using this program or any other weight-loss technique.

Calorie counting is important in most diet plans. Unfortunately, the process of looking up every item of food you eat is discouragingly tedious. And even if you conscientiously keep track of calories, how do you know how much progress you're making?

Your body burns a certain number of calories per day. The number depends on your sex, build, and activities. In order to lose weight, you must eat fewer calories than your body needs, forcing it to convert fat tissue into carbohydrates. On the other hand, if you eat more calories than your body burns in one day, the excess is converted into fat.

3500 Calories = 1 Pound

In order to lose one pound of fat, you have to miss 3500 calories. In order to gain a pound, you have to have an excess of 3500 calories. This is not on a daily basis; calories accumulate. So, if you ate 1000 more calories each day than your body used, you would gain one pound in about three and a half days.

Since any calculation is spread over many days, it can be hard to see progress, or to forecast how long it will take to shed excess weight. The computer is of great aid here.

"CalCalc" asks you a number of questions, such as your sex and age, to determine how many calories you need each day. You then enter everything you've eaten at the end of the day, selecting foods and quantities from a list (a *menu*, appropriately enough). Just press the letter corresponding to the food you ate. If you don't see a certain food, press RETURN to see more items.

Adding to the Menu

What if you ate a food not on the list? This is not too hard, since we've included only a sample selection of foods, found in the DATA statements from lines 1140 and up. To customize this list to your preferences and habits, just purchase a pocket-sized calorie counter (available at most grocery-store checkout counters). Then add to or change the DATA statements.

There is one DATA statement for each food. The first item on the line (after the word DATA) is the name of the food. Make the name less than 20 letters long. The next item, preceded with a comma, is the number of calories in an average serving, followed by a comma, and the description of the average serving, such as a 1 CUP or one 8" EAR. The last DATA statement (line 1500 here) should be END,0,0 which marks the end of the list.

After you've pressed the letter corresponding to the food you've eaten, the computer will display the quantity (such as one cup) and calories of an average serving. You enter the multiple or fraction in decimal of the quantity given. For example, if you drank two glasses of milk for breakfast, enter a 2, for two one-cup portions. If you had half a medium orange, enter 0.5. CalCalc then displays the calories for the food consumed, and the cumulative total of calories. You continue to enter foods for everything you've eaten.

Guesstimating

You can also approximate calories. For example, if you ate a chicken-filet sandwich, you could select T, chicken (one 4-ounce serving), and K, two one-slice portions of white bread. Or, if you can look on the wrapper of the product, you can enter the calories directly. Just press the number sign, #, instead of a letter, and enter the calories literally.

The Moment of Truth

After you've finished entering all the foods, the computer is ready to forecast weight loss. It bases this forecast on the assumption that you will eat about the same number of calories each day. Just enter the number of days you want to "look ahead," and CalCalc will tell you how much weight you will have lost. If you're eating too much, it will, with equal placidity, show you how much you'll have gained.

CalCalc makes dieting much easier. It goes beyond mere

automation of a calorie counter by letting you see the *effect* of changes. By cutting down on meals and checking your total calories with CalCalc, you can see if you'll lose weight.

CalCalc

```

100 GRAPHICS 0:POKE 752,1:POKE 82,0:GOSUB 10
20: DIM A$(1),FOOD$(19),AMOUNT$(10)
105 OPEN #1,4,0,"K"
110 PRINT "{DOWN} WARNING: CONSULT YOUR DOCTOR
    BEFORE"
120 PRINT "{9 SPACES}USING THIS PROGRAM OR A
    NY"
130 ? "{9 SPACES}OTHER WEIGHT-LOSS TECHNIQUE
    ."
140 ? "{DOWN}ARE YOU MALE OR FEMALE?"
150 GET #1,A:A$=CHR$(A):IF A$<>"M" AND A$<>"
    F" THEN 150
160 SX=0:IF A$="F" THEN SX=1
170 IF SX=0 THEN 200
180 ? "{DOWN}ARE YOU PREGNANT";:GOSUB 980:IF
    YES THEN PREG=1
190 ? "{DOWN}ARE YOU NURSING";:GOSUB 980:IF
    YES THEN NU=1
200 GOSUB 1020
210 ? "ENTER 0 IF NOT KNOWN:":?
220 TRAP 220:?"{UP}{DEL LINE}NUMBER OF CALO
    RIES CONSUMED?0{2 LEFT}";:POKE 752,0:INP
    UT CAL:POKE 752,1:TRAP 40000
230 IF CAL<0 THEN PRINT "{DOWN}{BELL} IMPOSSIB
LE";GOTO 200
240 IF CAL>=4500 THEN PRINT "{DOWN}";CAL;"CA
    LORIES? ARE YOU SURE";:GOSUB 980:IF 1-YE
    S THEN 200
250 IF CAL THEN 730
260 PX=0:PY=10:GOSUB 1020
270 FOR I=1 TO 26
280 READ FOOD$,CL,AMOUNT$
290 IF FOOD$="END" THEN 330
300 POSITION PX,PY:? CHR$(I+192);": ";FOOD$:P
    Y=PY+1
310 IF I=13 THEN PX=20:PY=10
320 NEXT I
330 REM
340 IF PEEK(20)>60 AND PEEK(20)<120 THEN POS
    ITION 2,23:? "ENTER 0 OR LETTER OF FOOD"
    ;
350 IF PEEK(20)>120 AND PEEK(20)<180 THEN PO
    SITION 2,23:? "PRESS RETURN TO GO ON
    {5 SPACES}";

```

```

360 IF PEEK(20)>180 THEN POSITION 2,23:?"PR
    ESS ■ WHEN DONE{4 SPACES}";:POKE 20,0
365 IF PEEK(764)=255 THEN 340
370 GET #1,A:A$=CHR$(A):IF (A$<"A" OR A$>"Z"
    ) AND A$<>CHR$(155) AND A$<>"*" AND A$<>
    "#" THEN 340
380 IF A$<>CHR$(155) THEN 410
390 NX=NX+1:IF FOOD$="END" THEN RESTORE :NX=
    0
400 GOTO 260
410 RESTORE
420 IF A$="#" THEN 600
430 IF A$="*" THEN 660
440 FOR I=1 TO NX*26+ASC(A$)-64
450 READ FOOD$,CL,AMOUNT$
460 NEXT I
470 GOSUB 1020
480 PRINT "FOOD: ";FOOD$
490 PRINT "CALORIES PER ";AMOUNT$;": ";CL
500 PRINT "{DOWN}ENTER QUANTITY OF ABOVE FOO
    D"
510 PRINT "CONSUMED, USING A MULTIPLE OR":?
520 TRAP 520:PRINT "{UP}{DEL LINE}A DECIMAL
    FRACTION?0{2 LEFT}";:POKE 752,0:INPUT QU
    :POKE 752,1:TRAP 40000
530 IF QU=0 THEN 590
540 IF QU<0 THEN PRINT "{DOWN}{BELL}IMPOSSIBLE
    ■":FOR W=1 TO 500:GOTO 470
550 PRINT "{DOWN}CALORIES OF ";FOOD$;": ";CL*
    QU
560 PRINT "{DOWN}CALORIES CONSUMED SO FAR:";
    :CAL=CAL+CL*QU:PRINT CAL
570 ? "{2 DOWN}PRESS RETURN TO CONTINUE..."
580 GET #1,A:A$=CHR$(A):IF A$<>CHR$(155) THE
    N 580
590 RESTORE :NX=0:GOTO 260
600 GOSUB 1020:?"{DOWN}ENTER ABSOLUTE QUANT
    ITY"
610 ? "{DOWN}OF CALORIES FOR FOOD NOT ON LIS
    T:":?
620 TRAP 620:?"{UP}{DEL LINE}?0{2 LEFT}";:P
    OKE 752,0:INPUT CL:POKE 752,1:TRAP 40000
630 IF CL=0 THEN NX=0:GOTO 260
640 IF CL<0 THEN ? "{DOWN}{BELL}IMPOSSIBLE":
    FOR W=1 TO 500:NEXT W:GOTO 600
650 QU=1:GOTO 560
660 GOSUB 1020
670 PRINT "TOTAL CALORIES CONSUMED: ";CAL

```

```

680 ? "{2 DOWN}DOES THAT SOUND REASONABLE";:
      GOSUB 980
690 IF YES THEN 730
700 ? "{DOWN}DO YOU WANT TO":? "RE-ENTER THE
      CALORIES";:GOSUB 980
710 IF YES THEN CAL=0:GOTO 260
720 PRINT "{CLEAR}":END
730 GOSUB 1020:?:?
740 TRAP 740:PRINT "{UP}{DEL LINE}WHAT IS YO
      UR AGE?20{3 LEFT}";:POKE 752,0:INPUT AGE
      :POKE 752,1:TRAP 40000
750 IF AGE<20 OR AGE>70 THEN PRINT "{DOWN}YO
      U MUST BE BETWEEN 20 AND 70"
760 IF AGE<20 OR AGE>70 THEN FOR W=1 TO 300:
      NEXT W:GOTO 730
770 IF AGE>=20 OR AGE<30 THEN CPD=3200:IF SX
      THEN CPD=2300
780 IF AGE>30 AND AGE<40 THEN CPD=3104:IF SX
      THEN CPD=2231
790 IF AGE>40 AND AGE<60 THEN CPD=2768:IF SX
      THEN CPD=1990
800 IF AGE>60 AND AGE<70 THEN CPD=2528:IF SX
      THEN CPD=1587
810 CPD=CPD+1000*NU+450*PREG
820 ? "{DOWN}ON A SCALE OF 1-5"
830 ? "1=MODERATELY ACTIVE, 5=VERY ACTIVE"
840 ? "HOW ACTIVE ARE YOU?"
850 GET #1,A:A$=CHR$(A):IF A$<"1" OR A$>"5"
      THEN 850
860 CPD=CPD+VAL(A$)*200
870 GOSUB 1020:?"{DOWN}ESTIMATED ENERGY EXP
      ENDITURE":?"IN CALORIES IN ONE DAY:";CP
      D
880 ? "{DOWN}TOTAL CALORIC INTAKE IN ONE DAY
      :";CAL
890 DF=CAL-CPD
900 ? "{DOWN}NUMBER OF DAYS TO PROJECTED"
910 TRAP 910:?"WEIGHT LOSS/GAIN?1{2 LEFT}";
      :POKE 752,0:INPUT ND:POKE 752,1:TRAP 400
      00
920 IF ND<1 THEN 910
930 ? "{DOWN}AT THE CURRENT CONSUMPTION, YOU
      SHOULD"
940 IF DF<0 THEN PRINT "LOSE ";:GOTO 960
950 ? "GAIN ";
960 PRINT INT(ABS(DF*ND)/3500);" POUNDS."
970 END
980 ? "? (Y/N):";

```

```

990 GET #1,A:A$=CHR$(A):IF A$<>"Y" AND A$<>"
  N" THEN 990
1000 YES=0:IF A$="N" THEN PRINT "NO":RETURN
1010 YES=1:?"YES":RETURN
1020 PRINT "{CLEAR}";
1030 ? " {3 N}{3 SPACES}{2 N}{3 SPACES}{N}
  {5 SPACES}{3 N}{3 SPACES}{2 N}
  {3 SPACES}{N}{4 SPACES}{3 N}"
1040 ? " {F}{G} {G} {F}{G} {G} {B}{2 G}
  {3 SPACES}{F}{G} {G} {F}{G} {G} {B}
  {2 G} {F}{G} {G}"
1050 ? "{B}{G}{H}{3 SPACES}{B}{G}{H}{J}
  {B} {B}{G}{H}{3 SPACES}{B}{G}{H}
  {J}{B} {B}{G}{H}{3 SPACES}"
1060 ? "{B} {3 SPACES}{B} {N}{V}{B}
  {B} {3 SPACES}{B} {N}{V}{B} {B}
  {B} {3 SPACES}{B} {G}{B} {B} {B}
  {3 SPACES}{B} {G}{B} {B} {B}"
1080 ? " {G}{2 M}{G}{B} {4 SPACES}{B} {M}
  {G} {G}{2 M}{G}{B} {4 SPACES}{B} {M}
  {2 G}{2 M}{G}"
1090 ? " {J}{3 SPACES} {G} {G} {G}
  {3 SPACES} {J}{3 SPACES} {G} {G}
  {G}{3 SPACES} {J}{3 SPACES}"
1110 ? :POKE 85,11:?"CALORIE CALCULATOR"
1120 PRINT "{40 R}"
1130 RETURN
1140 DATA CHEDDAR CHEESE,113,1" CUBE
1150 DATA COTTAGE CHEESE, 27, 1 OZ
1160 DATA WHOLE MILK,166,1 CUP
1170 DATA NONFAT MILK,87,1 CUP
1180 DATA GRAPEFRUIT,77,1 CUP
1190 DATA ORANGES,70,1 MED.
1200 DATA CANTALOUPE,37,1/2 MELON
1210 DATA APPLES,87,1 MED.
1220 DATA ORANGE JUICE,108,1 CUP
1230 DATA CORN FLAKES,96,1 CUP
1240 DATA WHITE BREAD,63,1 SLICE
1250 DATA WHOLE WHEAT BREAD,55,1 SLICE
1260 DATA HAMBURGER MEAT,316,3 OZ.
1270 DATA STEAK,293,3 OZ.
1280 DATA LAMB CHOP,480,4 OZ.
1290 DATA BACON,48,1 SLICE
1300 DATA HAM,340,3 OZ.
1310 DATA FLOUNDER,78,4 OZ.
1320 DATA TUNA FISH,170,3 OZ.
1330 DATA CHICKEN,227,4 OZ.
1340 DATA EGGS,640,1 CUP
1350 DATA SUGAR,48,1 TBS.

```

1360 DATA CARROTS,68,1 CUP
1370 DATA POTATOES,120,1 MED.
1380 DATA BEET GREENS,39,1 CUP
1390 DATA LETTUCE,7,4 SM. LEAVES
1400 DATA SPINACH,46,1 CUP
1410 DATA BAKED BEANS,295,1 CUP
1420 DATA LIMA BEANS,152,1 CUP
1430 DATA CORN,92,8" EAR
1440 DATA PEAS,74,.5 CUP
1450 DATA TOMATOES,30,1 MED.
1460 DATA 4% BEER,150,12 OZ.
1470 DATA BLACK COFFEE,9,1 CUP
1480 DATA COLA BEVERAGES,83,6 OZ.
1490 DATA POTATO CHIPS,108,10 2" CHIPS
1500 DATA END,0,0

Castle Quest

Timothy G. Baldwin

This entrancing, well-designed game offers you the best of both worlds. It has the drama, variety, and mystery of a good adventure game combined with the fast-paced excitement of an arcade game. Your job is to rid the kingdom of the three evil wizards. All this would be easy if the wizards weren't so zealously guarded by servants whose names reflect their personalities: bat-wingers, blinkers, chokers, crushers, and stompers.

You are in love with the Princess Dilayna and have asked her father the King for her hand in marriage. Her father does not particularly like you. He challenges you to demonstrate your worthiness by capturing the three evil wizards that have been ravaging the kingdom for years. They each live in their own castle protected by their servants—the bat-wingers, the blinkers, the chokers, the stompers, and the crushers. The castle rooms are rumored to be deadly, the untouchable walls, fast-moving enemies, and no exits. You reluctantly accept the King's challenge.

Fortunately, a friendly magician gives you a cloak that makes its wearer invisible. But the cloak's power works only for a limited time in each room. Once the time is up, you are instantly destroyed. The magician also gives you a magic spell that temporarily freezes all servants in a room. But you must use this spell with care: it will consume a portion of the cloak's power each time it is used.

Armed with these aids, you leave on your quest. The King wishes you good luck—or did he say good riddance?

The Three Wizards

The object of "Castle Quest" is to capture the three wizards. To reach each wizard, you must pass through the ten rooms of his castle. The rooms are inhabited by the wizard's servants, who move about quickly in an unpredictable manner. The higher numbered rooms in each castle have more servants (up to 32). The servants move progressively faster as you complete more rooms.

You have three lives to capture the first wizard. Capturing a wizard earns you three additional lives. Touching a servant or a room wall or failing to exit a room within the allotted time will

cause loss of a life. You cannot exit a room until you capture both door keys in that room by touching them. One key is invisible until the other key is touched.

Once both keys are captured, the room's exit appears—unless you are in a castle's tenth room. In this case, the wizard appears, and you must capture him before you can escape. Also, once you capture the first key, your presence becomes known to the wizard, and he causes room wall segments to move to block your escape. You must move quickly to avoid destruction.

Secret Passages

A counter at the top of the screen signals the amount of "cloak time" remaining. Pressing the joystick fire button will temporarily freeze the action, permitting you to move safely past a tight corner, but you lose 50 units of cloak time each time you use the freeze option. The room number and the number of your remaining lives are displayed at the top left of the screen. Your score—a measure of your ability to elude the many dangers involved—is displayed at the top right of the screen.

Room patterns, key locations, servant locations, and wizard placement are randomly generated, so be prepared to touch keys partially embedded in walls, move through weird mazes, etc. Sometimes a secret passageway is created at the screen bottom or in a room's right wall. You may use these passageways for a quick, easy escape.

Castle Quest

```

10 REM {5 SPACES} MEMORY SAVER{14 SPACES}
20 C0=0:C1=1:C2=2:C3=3:C4=4:C5=5:C6=6:C7=7:C
   B=8:C9=9:C10=10:C15=15:C16=16:C256=256:RA
   MTOP=PEEK(106):MISSION=C1
30 REM {7 SPACES}
40 GOSUB 1080:GOSUB 770:GRAPHICS C16:" "
   {CLEAR}":POKE 752,C1:SETCOLOR C2,C0,C0:GO
   SUB 310
50 T1=C8:GOSUB 1150:T1=C16:GOSUB 1150:G=C0:L
   =C3:Q=C0:C=C0:X1=C0:SCORE=C0
60 GOSUB 320
70 REM {4 SPACES} ROOM SETUP ROUTINE
   {9 SPACES}
80 GOSUB 970:GOSUB 450:GOSUB 1340:GOSUB 1500
   :POKE 1568,C1:POKE 77,0:POKE 53248,60:POK
   E 53249,W1
90 IF C=C10 THEN GOSUB 340

```

```

100 X=USR(1767):FOR I=C0 TO 100:NEXT I:POKE
    1568,F
110 REM {6 SPACES} MAIN PROGRAM LOOP
    {8 SPACES}
120 G=G-C1:IF (PEEK(1566)<>C0) OR (G<C0) THE
    N 400
130 IF PEEK(203)>204 THEN 520
140 POSITION 23-(G>999)-(G>99)-(G>C9),C0:? C
    HR$(B);G;CHR$(B):IF G<100 THEN SETCOLOR
    C2,C4,C0
150 X=PEEK(53260):IF (X-X1)>=C2 THEN POKE 53
    250,W2:POKE 53249,C0:IF PEEK(706)<>N THE
    N GOSUB 380:POKE 706,N
160 IF X-X1>=C4 THEN POKE 53251,W3:POKE 5325
    0,C0
170 IF X>=C6 THEN GOSUB 260
180 IF STRIG(C0)=C0 THEN POKE 1568,C1:G=G-50
    :FOR I=0 TO 250:NEXT I:POKE 1568,F
190 CHBASE=RAMTOP-C8-C8*(INT(G/2)=G/2):POKE
    756,CHBASE
200 IF PEEK(706)=N THEN IF RND(C0)>.95 THEN
    PLOT INT(RND(C0)*38),INT(RND(C0)*22):GO
    SUB 240
210 IF STICK(C0)<>15 THEN SOUND C2,100,C6,C8
    :SOUND C2,C0,C0,C0
220 GOTO 120
230 REM {3 SPACES} "SHOOTING" SOUND ROUTINE
    {3 SPACES}
240 FOR I=C0 TO 30:SOUND C0,I,C0,C15:NEXT I:
    SOUND C0,C0,C0,C0:RETURN
250 REM ROOM EXIT OPENING ROUTINE
    {4 SPACES}
260 IF C=C10 THEN IF X<>14 THEN RETURN
270 FOR I=C0 TO C5:POKE SC+C10*40+I*40-C1,C0
    :NEXT I:POKE 53278,255:FOR I=C15 TO C0 S
    TEP -C1:SOUND C0,C10,C10,I
280 SOUND C1,11,C10,I+C1:SOUND C2,12,C10,I+C
    2:SOUND 3,13,10,I+3:NEXT I:FOR I=0 TO 3:
    SOUND I,C0,C0,C0:NEXT I
290 POKE 53251,C0:POKE 53250,C0:POKE 53278,2
    55:RETURN
300 REM USER INFORMATION ROUTINES
    {3 SPACES}
310 POSITION C10+C1,C10:? "Wait for game set
    up":RETURN
320 C=C+C1:POSITION C10,C10:? "Get ready for
    Room ";C:C=C-C1:RETURN
330 REM WIZARD PLOTTING ROUTINE{6 SPACES}

```

```

340 PL=(RAMTOP-9)*256:PL=PL+52+INT(RND(C0)*1
51):RESTORE 350:FOR I=C0 TO 11:READ Z:PO
KE PL+I,Z:NEXT I
350 DATA 102,36,126,90,126,126,66,90,60,60,3
6,102
360 W3=70+INT(RND(C0)*130):POKE 707,P:RETURN

370 REM "KEY TOUCHING" SOUND ROUTINE
380 SOUND C2,20,C10,C10:SOUND C1,80,C10,C10:
FOR I=0 TO 30:NEXT I:SOUND C1,C0,C0,C0:S
OUND C2,C0,C0,C0:RETURN
390 REM USER FAILS TO ESCAPE ROOM
{3 SPACES}{8 SPACES} ROUTINE(21 SPACES)
400 FOR I=C0 TO C3:POKE 53248+I,C1:NEXT I:PO
KE 1568,C1:?"{CLEAR}":SETCOLOR C2,C0,C0
:IF Q THEN RETURN
410 POKE DL+C15,C7:POSITION C4,C10:IF Q THEN
RETURN
420 POKE 756,224:?"TOUGH LUCK!":FOR I=C0 TO
200:SOUND C0,C6,100,C8:NEXT I:SOUND C0,
C0,C0,C0:T2=C1
430 POKE DL+C15,C2:L=L-C1:?"{CLEAR}":C=C-1:
GOSUB 320:C=C+1:GOTO 80+500*(L<=C0)

440 REM DETERMINE NEXT ROOM'S(9 SPACES)
{8 SPACES}CHARACTERISTICS ROUTINE
{7 SPACES}
450 A=INT(C16*RND(C0))*C16+C6:M=INT(C16*RND(
C0))*C16+C2:N=INT(C16*RND(C0))*C16+C4:P=
INT(C16*RND(C0))*C16+C8
460 B=33+C-C6*(C>5):C=C+C1:D=C2+C2*(C>C1)+C4
*(C>C3)+C8*(C>C6)+C16*(C>C9)
470 E=INT(RND(0)*5+7):POKE 1763,E
480 F=C2+(C>C9)+C2*(MISSION-C1)
490 G=100+C*50:COLOR B:POKE 1578,31:POKE 156
6,C0:POKE 756,RAMTOP-C8:POKE 53278,255:X
1=C0
500 SETCOLOR 2,C7*(C=7)+C2*(C=8)+C1*(C=9)+C3
*(C=10),C0:RETURN
510 REM USER ESCAPES FROM A ROOM(5 SPACES)
{8 SPACES} ROUTINE(22 SPACES)
520 Q=C1:GOSUB 400:GOSUB 410:POKE 756,224:?"
{3 SPACES}ATTABOY!":Q=C0
530 FOR I=C0 TO C5:SOUND C0,C10,50,C8:POKE 7
05,C10:POKE 706,C10:POKE 710,C10:POKE 71
2,C10:FOR J=C0 TO 50:NEXT J
540 SOUND C0,C10,100,C8:POKE 705,C0:POKE 706
,C0:POKE 710,C0:POKE 712,C0:FOR J=C0 TO
50:NEXT J:NEXT I

```

```

550 SOUND C0,C0,C0,C0:POKE DL+C15,C2:? "
    {CLEAR}":GOSUB 320:SCORE=SCORE+MISSION*INT((G*C)/C10)
560 IF C=C10 THEN GOTO 580+110*(MISSION=C3)
570 GOTO 80
580 REM {3 SPACES} END A QUEST ROUTINE
    {8 SPACES}
590 ? "{CLEAR}":POKE DL+C9,C6:POKE DL+11,C6:
    POKE DL+13,C6:POKE DL+15,C6:POKE 707,C0:
    IF L<=C0 THEN 660
600 POSITION C2,C4:? "congratulations!":POSITION 26,C5:? "YOU HAVE":POSITION C3,C7:?
    "COMPLETED YOUR"
610 POSITION 27,C8:? "QUEST":C=C0:L=L+C3
620 POSITION C5,15:? "Press START to continue":POSITION C5,17:? "Press SYSTEM RESET
    ■ to quit"
630 POSITION C5,19:? "SCORE: ";SCORE
640 POKE 53279,C8:IF PEEK(53279)<>C6 THEN 640
650 ? "{CLEAR}":POKE DL+C9,C2:POKE DL+11,C2:
    POKE DL+13,C2:POKE DL+15,C2:MISSION=MISSION+(L>C0)*C1:GOTO 60+620*(L<=C0)
660 POSITION C7,C4:? "SORRY!":POSITION 24,C5:
    ? "you blew it.":POSITION C2,C7:? "quests completed ";MISSION-C1
670 GOTO 620
680 RUN
690 REM USER WINS THE GAME ROUTINE!
    {3 SPACES}
700 GRAPHICS 2:SETCOLOR C2,C0,C0:POSITION C6,C4:? #6;"YOU WON!":? "Press SYSTEM RESET
    ■ and then 'RUN' to";
710 POKE 752,1:? :? "begin a new game."
720 POSITION C1,C7:? #6;"final score ";SCORE
730 FOR I=255 TO C0 STEP -C1:SOUND C0,I,10,10:POKE 712,I:POKE 710,I:NEXT I
740 GOTO 740
750 POKE 1568,C1:RUN
760 REM PUT A VERTICAL BLANK INTERRUPT
    {8 SPACES} ROUTINE IN PAGE 6 OF MEMORY
770 RESTORE 790:FOR I=1536 TO 1536+247:READ A:POKE I,A:NEXT I
780 RETURN
790 DATA 173,4,208,201,4,240,2,208,22,173,99,228,141,36,2
800 DATA 173,100,228,141,37,2,141,30,6,141,30,208,76,98,228

```

```

810 DATA 0,162,2,202,240,42,138,72,173,10,21
    0,41,7,10,170
820 DATA 189,0,1,133,206,133,208,232,189,0,1
    ,133,207,133,209
830 DATA 32,148,6,165,207,157,0,1,202,165,20
    6,157,0,1,104
840 DATA 170,208,211,162,5,173,120,2,202,240
    ,197,24,106,176,249
850 DATA 72,224,2,240,8,224,1,208,13,230,203
    ,208,2,198,203
860 DATA 165,203,141,0,208,208,32,169,0,224,
    4,240,8,168,145
870 DATA 204,230,204,76,134,6,160,7,145,204,
    198,204,160,0,185
880 DATA 240,6,145,204,200,192,8,208,246,104
    ,76,83,6,160,0
890 DATA 152,145,206,173,10,210,41,1,208,15,
    169,56,141,201,6
900 DATA 169,233,141,204,6,141,210,6,208,13,
    169,24,141,201,6
910 DATA 169,105,141,204,6,141,210,6,173,10,
    210,41,1,208,2
920 DATA 169,40,141,205,6,216,0,165,206,0,0,
    133,206,165,207,0
930 DATA 0,133,207,177,206,240,8,165,208,133
    ,206,165,209,133,207
940 DATA 169,11,145,206,96,104,168,162,6,169
    ,7,76,92,228,60
950 DATA 126,90,126,90,102,126,60
960 REM SETUP PLAYER-MISSILE GRAPH-
    (9 SPACES) TCS ROUTINE (18 SPACES)
970 POKE 559,62:POKE 54279,RAMTOP-C16:POKE 5
    3248,C1:POKE 53277,C3
980 PL=RAMTOP-12:Y=PEEK(88):Z=PEEK(89):POKE
    88,C0:POKE 89,PL:POKE 106,PL+C3:?"
    (CLEAR)":POKE 88,Y:POKE 89,Z
990 POKE 106,PL+12:PL=PL*C256+120:IF C=C0 OR
    C=C10 THEN Z=(RAMTOP-C9)*C256:FOR I=Z T
    O Z+255:POKE I,C0:NEXT I
1000 FOR I=C0 TO C7:POKE PL+I,PEEK(1776+I):N
    EXT I
1010 POKE 203,60:POKE 204,PL-INT(PL/C256)*C2
    56:POKE 205,INT(PL/C256)
1020 PL=(RAMTOP-11)*C256:PL=PL+52+INT(RND(C0
    )*151):RESTORE 1030:FOR I=C0 TO C7:READ
    Z:POKE PL+I,Z:NEXT I
1030 DATA 0,6,15,249,255,166,160,0
1040 W1=70+INT(RND(C0)*130):PL=(RAMTOP-C10)*
    C256:PL=PL+52+INT(RND(C0)*151):RESTORE
    1030:FOR I=C0 TO C7

```

```

1050 READ Z:POKE PL+I,Z:NEXT I:W2=70+INT(RND
(C0)*130):POKE 705,M:IF T2=C1 THEN C=C-
C1:T2=C0
1060 POKE 53249,C0:POKE 53250,C0:RETURN
1070 REM {4 SPACES} TITLE PAGE ROUTINE
{7 SPACES}
1080 GRAPHICS 18:SETCOLOR C2,C0,C0:POKE 708,
202:POSITION C5,C2:? #C6;"CASTLE":POSIT
ION C9,C4:? #C6;"QUEST"
1090 DL=PEEK(560)+C256*PEEK(561):POKE DL+13,
C2
1100 POSITION C3,C8:? #C6;"How many rooms ca
n you survive?"
1110 FOR I=C0 TO C3:POKE 708,C0:SOUND C0,60,
C10,C8:FOR J=C0 TO 100:NEXT J:SOUND C0,
160,C10,C8:POKE 708,202
1120 FOR J=C0 TO 100:NEXT J:NEXT I
1130 SOUND C0,C0,C0,C0:RETURN
1140 REM SETUP SPECIAL CHARACTER SETS
{9 SPACES} ROUTINE(22 SPACES)
1150 RESTORE 1160:CL=(RAMTOP-T1)*C256:FOR I=
CL+C8 TO CL+95:READ A:POKE I,A:NEXT I
1160 DATA 204,51,204,51,204,51,204,51,102,15
3,102,153,102,153,102,153
1170 DATA 136,34,136,34,136,34,136,34,68,17,
68,17,68,17,68,17
1180 DATA 36,146,73,36,146,73,36,146,255,255
,255,255,255,255,255,255
1190 DATA 195,102,60,24,24,0,0,0
1200 DATA 255,255,195,195,195,195,255,255
1210 DATA 255,255,0,0,0,0,255,255
1220 DATA 24,24,60,24,255,199,199,255
1230 DATA 24,255,0,0,0,0,0,0
1240 FOR I=128 TO 224:POKE CL+I,PEEK(57344+I
):NEXT I
1250 DL=PEEK(560)+C256*PEEK(561):IF T1=C16 T
HEN RESTORE 1260:FOR I=CL+56 TO CL+95:R
EAD A:POKE I,A:NEXT I
1260 DATA 0,0,0,24,24,60,102,195
1270 DATA 0,0,60,60,60,60,0,0
1280 DATA 0,0,255,255,255,255,0,0
1290 DATA 60,24,24,24,60,60,0,0
1300 DATA 24,24,24,24,24,24,24,255
1310 IF T1=C16 THEN FOR I=CL TO CL+C7:POKE I
,C0:NEXT I
1320 RETURN
1330 REM RANDOM ROOM MAZE GENERATOR
{9 SPACES} ROUTINE(20 SPACES)
1340 ? "{CLEAR}":POKE 752,C1

```

```

1350 PLOT C0,C0:DRAWTO 39,C0:DRAWTO 39,23:DR
    AWTO C0,23:DRAWTO C0,C0
1360 X=C10:Y=C0:Z=C7:GOSUB 1400:X=C15:Y=C5:Z
    =13:GOSUB 1400:X=C10:Y=C16:Z=C7:GOSUB 1
    400
1370 IF RND(C0)<0.5 THEN PLOT RND(C0)*31+C8,
    11:DRAWTO RND(C0)*31+C8,11
1380 POSITION C6,C0:? C:POKE 704,A:POKE 705,
    M
1390 POSITION C9,C0:? L:POSITION 30,C0:? SCO
    RE:RETURN
1400 ON INT(RND(C0)*C8+C1) GOSUB 1410,1420,1
    430,1440,1450,1460,1470,1480
1410 RETURN
1420 PLOT X,Y:DRAWTO X,Y+Z:RETURN
1430 X=X+C10:GOSUB 1420:RETURN
1440 X=X+20:GOSUB 1420:RETURN
1450 GOSUB 1420:GOSUB 1430:RETURN
1460 GOSUB 1430:GOSUB 1430:RETURN
1470 GOSUB 1420:GOSUB 1460:RETURN
1480 POP :GOTO 1360
1490 REM WIZARD'S SERVANTS PLOTTING
    {3 SPACES}{9 SPACES}AND ADDRESS CALCULA
TION ROUT-{9 SPACES}TNE. ADDRESSES KEPT
IN STACK
1500 SC=PEEK(88)+C256*PEEK(89):FOR I=C0 TO D
    -C1:IF INT(RND(C0)*C4)>C2 THEN 1520
1510 H=SC+40+INT(RND(C0)*279):GOTO 1530
1520 H=SC+680+INT(RND(C0)*239)
1530 HI=INT(H/C256):LO=H-HI*C256:POKE C256+I
    *C2,LO:POKE H,E
1540 POKE C256+I*C2+C1,HI:NEXT I:IF D=32 THE
    N RETURN
1550 FOR I=(D-C1) TO 31:POKE C256+I*C2+C1,25
    4:NEXT I:RETURN

```


Scriptor: An Atari Word Processor

Charles Brannon

"Scriptor" is an easy-to-use, full-scrolling, character-oriented, multifunction word processor, requiring an Atari 800XL or 400/800 with a minimum of 32K of memory (40K recommended), an Epson MX-80 or Atari 825 printer, and an Atari 810 disk drive. It is programmed in both BASIC and machine language. For instructions on typing in the program, see the section under Typing It In.

Through the Ruby

Computers don't just calculate with numbers—they can also work with text. Five-inch disks can replace stacks of files. Computers can sort, search, select, and update any kind of information. They can *focus* information. In this sense, the computer is like the ruby crystal in a laser. Ordinary random light waves are transformed and concentrated through the ruby into a tight, powerful beam. Computers can do the same for information.

Word Processing

Electronic text is more "liquid," easier to work with, than words solidified on paper (*hard copy*). This is what makes word processing special: the extraordinary editing power it gives you. Distinctions between a rough draft and a final draft are meaningless; the work is typed, changed dynamically, and stored to disk. It can then later be recalled, revised, and printed out. Very little retyping is necessary. What a boon for anyone who writes.

Converts to word processing immediately notice an improvement in their writing. The entire manuscript becomes "alive," not committed to paper. Changing a word or a sentence, inserting a line or a paragraph are all accomplished with ease. For example, take just one key, the backspace key (called RUBOUT on some computers or terminals). When this key is struck, the last character typed is erased from the screen. Compare this to the frequently elaborate typewriter correction schemes.

Besides the disk file, which has already been mentioned and

which will be explained in greater detail later, an important concept in word processing is the *cursor*. Named after the clear plastic slide on a slide rule, the cursor shows you where the next character you type is going to appear. It usually looks like an underline, , or a solid square. Users familiar with any computer have already encountered the cursor. The computer itself doesn't need a cursor; but since you can type anywhere on the screen, the cursor is vital so that you can know where you are.

The cursor can be moved up, down, left, and right with special keys, usually with arrows on them. To correct the following line:

The quick brown dox jumped■

you would either press backspace ten times, erasing the text as you go, or press cursor-left ten times. The cursor moves over the characters without erasing them. It is then resting on the d:

The quick brown d|ox jumped

You can correct the error by typing *f*, which overstrikes (replaces) the *d*.

The quick brown f|ox jumped

The cursor can then be moved to the end of the line (ten cursor-rights), and typing resumed.

This sounds harder than it really is. Cursor editing becomes second nature after only hours of use. The cursor UP/DOWN keys can reach lines of text above and below the current line. It is like rolling a typewriter's platen up or down, but with one important difference—the "paper" is one continuous, long sheet.

Getting Specific

Two very special functions are *insert* and *delete*. Insert lets you add text in the middle of a line, by pressing INSERT to insert spaces in the text, and then typing in the word. For example:

To be or to be, that is the question.■

The cursor is placed on the second *to*, and INSERT is pressed four times (three for n-o-t, and one for a space):

To be or ■ to be, that is the question.

The word *not* is then typed:

To be or not■to be, that is the question.

Delete is used to erase text. As distinguished from mere back-spacing or spacing over a word, delete closes up the space after the deleted word.

Take out a word

Take Out a word

1. (cursor is moved to "o")

Take ut a word

2. (DELETE typed; "o" disappears, "ut a word" moves left.)

Take a word

(DELETE is typed four times.)

Insert and delete can also act on words, sentences, lines, or entire paragraphs in a similar way.

Disk Files

A file is simply a permanent record of your text. When the computer's power is turned off, it forgets everything except what is "burned" (in ROM memory) into it permanently. Your text is obviously not "burned in," or you couldn't ever change it. If you have a blackout, or a fuse blows, say good-bye to your text.

Catastrophes aside, you certainly don't want to leave your computer on all the time, or keep the computer tied up with your text forever. Fortunately, you can save your text on disk, ready for any later revisions. You can type it one time, save your text, and print it out when convenient.

Since a disk can store more than one document (unless it's very long), you and the computer must have some way to distinguish and separate one file from another. This is usually done via a *directory*, a list of filenames. You access a file by giving the computer the file's name.

"Scriptor," the word processor program at the end of this article, has many features usually found only in professional word processors, but it lacks a few features such as search and replace, justification, data base merge, etc. Also, it is written in BASIC, so it can be rather slow at times. It is, however, aided by several machine language subroutines for time-critical situations such as disk input/output and some editing features.

Typing It In

Program 1 is the Scriptor program itself. Type it carefully, since it contains many critical machine language DATA statements. Extra

time spent in typing it in will reward you with a smoother, bug-free word processor. Remember to use the Listing Conventions. Use the Atari logo key to enter inverse video.

To give you more memory for text, Scriptor deletes a substantial portion of itself after it initializes (sets up). Don't worry—the program is busy running while the screen flashes; it just takes awhile. The setup lines from 5000-6999 are automatically erased.

If you quit the program and try to run it again, the program will automatically try to re-RUN itself anew from disk. If you've changed disks, you'll need to reload it yourself. You should SAVE the program with the filename "D:SCRIPTOR" or change line 455 appropriately. Be sure to SAVE Scriptor after you've typed it, before you run it, or you will find a sizeable chunk of your typing erased when you exit. You can free up more memory for text by deleting the "help" function. Take out all lines from 1570 to 1700 and remove line 775. If you'd rather keep this handy aid, leave these lines alone.

If you get the message "Error in DATA statements" when you run the program, you need to check your typing of the machine language DATA statements at the end of the program. Also make sure you haven't typed a letter O for a zero (the zero is thinner than the O).

If you have an Atari 825 printer, you will need to type in the lines in Program 2. This will replace the lines used for the MX-80 with lines applicable to the 825 80-column printer. If you have another printer, refrain from using special characters such as underlining, and you will probably be able to get one of the sets of lines to work.

Getting Started

Scriptor is a full-scrolling, character-oriented word processor. The use of cursor control keys is similar to normal Atari editor functions, with these exceptions.

I. <RETURN> is used only to force a carriage return, as at the end of a paragraph, or to print a blank line. The computer will format your line when you print it out, so just type continuously. Do not press <RETURN> at the end of each line. Pressing <RETURN> prints a back-arrow at the end of the line, and erases all text to the end of that line.

II. Insert and Delete character (CTRL-INSERT/CTRL-DELETE) work on whole "paragraphs." A paragraph is a block of lines from the cursor to a "back-arrow." If there is no back-arrow,

one is assumed at the end of text. Therefore, Insert and Delete can be quite slow if you don't have a back-arrow somewhere.

III. Insert and Delete line work on the entire document. The screen will blank during this operation. This is normal and speeds up the process, as it can be slow on long documents.

IV. All TAB controls work normally, just a little slower. <CTRL-K> will clear all tab settings.

V. <CLEAR> will not clear the screen. It is used to erase all or part of the text. Press <CLEAR> <A> to erase all text. Press the Atari logo key to abort the erase function.

VI. The break key is disabled. Use <CTRL-Q> to exit the program.

VII. The ESC key enters the "mini-DOS." (See below.)

VIII. The console keys are "live"; see a description of their functions later.

IX. The Atari logo key is disabled for normal typing. Within prompts, it acts as an "abort" key.

Getting Control

Since the Atari is not a dedicated word processor (that means it's not just a "word processing machine" like a Lanier, but is, rather, a general-purpose computer), it does not have special keys to activate word processing functions. Instead, the <CTRL-key> combination is used. For example, to quit the program, you would hold down <CTRL> and press <Q>. The CTRL key stands for "Control"—it is like a special shift key. The keys are linked *mnemonically* (easy to remember) to the commands they stand for, such as <P> for Print Text. To get a list of the commands and what they stand for at any time, just press <CTRL-?> (hold down CTRL and press the question mark) for a HELP menu. See Table 1 for a quick-reference chart of the commands.

Going Around the Block

An important feature in a word processor is block move and delete. Scriptor lets you define a series of up to 23 lines. You can then move these lines to another place in the text with Line Duplicate, or delete the defined lines with <CLEAR/D> (Erase: Defined lines). To define a block of lines, just place the cursor on the first line and press <CTRL-D>. A flashing arrow will appear to the left of the line. Press cursor-down, and another symbol will appear underneath. Press cursor-down until all the desired lines

have an arrow to their left. Then press <RETURN>. If you make a mistake, just try again, or press cursor-up while defining.

To copy these lines to another place, position the cursor at the place you want the lines to appear, and press <CTRL-L>. If you haven't defined any lines, this command will be ignored. Note that you can press this key more than once to make many copies of the lines. You may want to delete the defined lines after you move them. Press <CLEAR>. You will see the prompt "ERASE:". Press <D>. The lines will be deleted, just as if you used Delete line multiple times.

A Mini-DOS

The ESC key activates the mini-DOS. It lets you look at the directory and scratch, rename, lock, or unlock files. When you press <ESC>, you will see:

Directory, **L**ock, **U**nlock, **R**ename, **S**cratch?

Press the appropriate key. For all except the directory, you will need to enter a filename. The cursor, a half box, will be at the top of the screen. The only editing key you can use here is backspace.

Remember that you can abort any time before pressing <RETURN> by pressing the logo key. While the directory is listed, you can press <ESC> again to keep the directory on the screen while you use one of the other functions. You can also press [SELECT] (see later) to save or recall a file while looking at the directory. If you get an error message at the top of the screen, check the disk and your entry and try again.

For the Record . . .

To save or recall a document, press [SELECT]. The screen will display:

Save or **R**ecall

Press the appropriate key, enter the filename, and the document will either be stored or retrieved. If you Recall a document, it loads starting at the line the cursor is on. This lets you add text to a document. Press START twice to home the cursor to the start of the text. If you get an error message, check to see you have the right disk, consult the *DOS Manual*, and try again. Remember that your filename must start with a capital letter and be followed by up to seven capital letters or numbers. You can optionally put a three-character extension on the file if you separate it with a

period, for example, EDITOR. DOC, DRAFT3.CGB, DUNGEON. MAP, etc. *You should not enter the "D:" prefix.*

Printer a la Mode

Different printers offer special print densities and formats such as boldface, underlining, super- and subscripts, double-width, condensed, proportional spacing, etc. To underline a word or phrase, enclose it in <CTRL-brackets>. In other words, <CTRL-,> is underlining on, and <CTRL-.> is underlining off. Underlining works only on the 825 printer. If you have GRAF-TRAX installed in your MX-80, underlining produces italics.

The following is an advanced technique. You can define up to ten special characters and print them at any spot in your text. To define a character, set up a format line (see the discussion of format lines, below) with <CTRL-F> and enter your definitions such as 1 = 123:2 = 125:3 = 27, etc. You can then output the CHR\$ code of the defined characters by embedding a caret ("^") in your text, followed by the number (for example, ^4). If you don't put a number after it, a caret will print; otherwise, the character associated with the number (0-9) will be output. You can also output ASCII characters from within a format line with the "as" format command. For example, "as27:as18" will activate proportional spacing on the 825 printer. Use "as27:as69" for emphasized mode on the MX-80.

Formatting Text

Since you are typing in the raw text, with no margins or line breaks, how does the computer print a nice formatted page? The computer assumes a left margin of 5, a right margin of 75, single spacing, a page length of 66, and 50 lines to be printed per page. You can change these default values with a format line.

A format line is like an embedded command line. The line starts with a format character to prevent the line from being printed out. To get the format character, press <CTRL-F>. You should get a right-pointed wedge. Then type in your commands. All commands are two lowercase letters, usually followed by a number. You can put multiple commands on the same line if you separate them with colons. For example, the following line:

```
▶1M10:RM70:SP2←
```

will set the left margin to ten, the right margin to 70, and line spacing to two. Here is an explanation of each formatting command. Also see Table 2 for quick reference.

*Note that **n** represents a number, with no space between the command and the number. No real error-checking is performed on the number.*

- as***n* Sends byte *n* to printer.
- cm**: Comment line. You can type one screen line of comments. They will not be printed to the printer. They are just for your convenience.
- cn***n* Centering. If *n* = 1, then centering will be *on*, and all following lines will be centered until reset by **cn**0. If *n* = 0, then centering is turned *off*.
- fp** Forced paging. Normally, the printer will page, or go on to the next page, when the number of lines printed equals your lines per page (**lp**), which defaults to 50. Forced paging pages to the next page, regardless.
- lm***n* *n* = left margin, which should be less than the right margin.
- ln***n* Prints *n* blank lines.
- lp***n* Sets lines per page to *n* — *n* should be less than the page length, to allow some blank space at the bottom of each page.
- nf**: *filename* Will chain to next specified file, permitting a document to be split up into many parts. The **nf** insures that they will all print as one big file. The formatting commands carry over to each file.
- pl***n* Sets the page length, which is almost always (and defaults to) 66.
- rm***n* *n* = right margin, which should be less than the maximum width and greater than the left margin.
- sp***n* *n* = 1, single spacing; *n* = 2, double spacing; *n* = 3, triple spacing; etc.

Start the Presses

To print your document, press <CTRL-P>. You should see:

PRINT: (C/F)

To start printing, just press <RETURN>. The printer head should be positioned at about the start of the page. The C/F indicates any selected option. C stands for Continuous Print. You would use this option with pinfeed or roll paper. It will automatically page to the start of each sheet. If you do not select continuous print, the computer will beep at the end of each page and pause. You should put in another sheet of paper and press <RETURN> to continue printing.

Note that pressing a key any other time during printing will abort the printout. The F option stands for Fast Printout. It will blank the screen during the printing, increasing printing speed better than 30 percent. Some people, however, find a blank screen disconcerting. To select one of the options, press either C or F. The appropriate letter will light up and flash. To reset the option (cancel it), press the key again. Press < RETURN > when you are ready to print the text.

Customizing Scriptor

The program is fairly well-structured, with separate sections for all functions. The control keys are executed via a branching IF/THEN "bucket brigade." Just patch in your own command where desired. Some functions you may want to add are block transfer (performs both block insert and block delete), Search and Replace, Insert from Disk, and simple data merge. Machine language programmers may want to try their hand at speeding up certain aspects of the program, such as Insert Line, Delete Line, and even Print Text.

Here are some other useful subroutines. GOSUB 540 returns the number of lines the user has typed (not necessarily the maximum number of lines) in EOT. GOSUB 600 clears the top line of the screen and positions the cursor at the first character, ready for a message. GOSUB 460 performs error-checking and adjustments on the X-Y position of the cursor. GOSUB 2650 returns an adjusted (uppercase if AL = 1, no cursor controls, etc.) character in A. GOSUB 2730 is a pseudo-INPUT routine that returns IN\$. Variable MX controls the maximum number of characters.

TRAP 2170 will vector errors to an I/O Error message. There are two reentry points for the editor proper: GOTO 650, which clears and "refreshes" the screen, and GOTO 680, which just adjusts the cursor and continues keyboard entry (faster).

Primary variables are: CL—the pointer to the top line (from 0-#lines) of the screen; X—the horizontal position of the cursor 2-39; Y—the vertical position of the cursor on the screen, 1-23; TX\$—the string that contains all the text and is organized in 38 character substrings, one for each line; T\$ and T—"temporary variables"; A—usually a keystroke typed; SCR—the address of the screen memory origin; NL—number of defined lines; FRL—the starting line in text of the defined lines; RL—the starting line in TX\$ for reserved lines (the buffer). Several constants are Q0, Q1, Q23—which return 0, 1, or 23 (saves memory); L2 = 38; L = 40; B\$ is 38 null (CHR\$(0)) characters.

Changes for the 800XL and 1200X

Scriptor as originally printed would not run on an XL model. The modifications for the 1200XL are contained in Program 3 and for the 800XL in Program 4. Simply substitute and/or add the lines to the main listing, Program 1.

There is another problem which might result from running Scriptor on an XL. Scriptor, as mentioned before, deletes part of itself. The deletion of lines will sometimes cause Atari BASIC to lock up. Be sure to include line 7000, even though it is just a REM statement: line 7000 will help prevent the lock-up.

If Scriptor still locks up, you will have to experiment. Try adding a REM statement to the end of one of the lines at the end of the program (6000–6060). What you are trying to do is change the length of the lines being deleted.

Table 1. Editing Commands

Control Keys

A Advance one screen forward
B Back up one screen
D Define lines
F Print format character
G Go to specified line
K Clear all tab settings
L Duplicate defined lines
P Print document
Q Quit program

SHIFT-INSERT
SHIFT-DELETE
CTRL-INSERT
CTRL-DELETE
CLEAR

CAPS/LOWR
ESC

Cursor keys

[OPTION]

[SELECT]

[START]

[CTRL-,]

[CTRL-.]

x

Insert a line

Delete a line

Insert a space

Delete a character

Erase:

A = All R = Remainder

D = Defined lines

Upper-or lowercase

Mini-DOS

Moves cursor with two-way scrolling

Nondestructive carriage return

Save or Recall text

"Home" cursor

Underlining on

Underlining off

Print special character

Table 2. Formatting Commands

<u>Command</u>	<u>Description</u>	<u>Default</u>
asn	Send ASCII character <i>n</i> to printer	—
cm:xxxx	Comment line	—
cn <i>n</i>	Centering: 1 = on, 0 = off	0 Off
fp	Forced Paging	—
lm <i>n</i>	Set left margin to <i>n</i>	5
ln <i>n</i>	Do <i>n</i> linefeeds	—
lp <i>n</i>	Set lines per page to <i>n</i>	50
nf:file	Link to Next File	—
pl <i>n</i>	Page length	66
rm <i>n</i>	Set right margin to <i>n</i>	75
sp <i>n</i>	Set line spacing	1 (single)

Program 1. Scriptor

```

100 REM SCRIPTOR WORD PROCESSOR
110 GOTO 5000
455 RUN "D:SCRIPTOR"
460 PF=Q0: IF X<2 THEN X=39:Y=Y-Q1
470 IF X>39 THEN X=2:Y=Y+Q1
480 IF Y<Q1 THEN Y=Q1:CL=CL-Q1:PF=Q1
490 IF Y>Q23 THEN Y=Q23:CL=CL+Q1:PF=Q1
500 IF CL<Q0 THEN CL=Q0
510 IF CL>(MXL-Q23) THEN CL=MXL-Q23
520 IF PF=Q0 THEN RETURN
530 LOC=CL*L2+Q1:T=USR(SCRZAP,ADR(TX$(LOC)))
   :RETURN
540 REM *** FIND END OF TEXT
550 P=ADR(TX$):T=P+RL*L2-Q1
560 A=USR(EDCOM,T,P,2):A=A-P
570 LC=A:EOT=INT(A/L2)
580 RETURN
590 REM *** ERASE TOP LINE
600 COLOR 32:PLOT Q1,Q0:DRAWTO L2,Q0:PLOT Q1
   ,Q0:RETURN
610 REM *** START OF EDITOR
611 MXL=INT(FRE(Q0)/40)-25:RL=MXL+1
612 DIM TX$((MXL+Q23)*L2):? CHR$(125);
613 TX$=CHR$(Q0):TX$((MXL+Q23)*L2)=TX$:TX$(2
   ')=TX$
620 SCR=PEEK(88)+256*PEEK(89):POKE 559,46:PO
   KE 842,12
630 X=2:Y=Q1:CL=Q0:POKE 702,Q0
640 REM *** ENTRY FOR EACH PAGE
650 POKE 54286,192

```

```

655 POSITION Q0,Q0:? "{7 SPACES}Scriptor Wor
d Processor";:COLOR 32:DRAWTO L2,Q0:PLOT
32,Q0
660 LOC=CL*L2+Q1:T=USR(SCRZAP,ADR(TX$(LOC)))
670 IF TF THEN TF=Q0:GOTO 810
675 IF FIRST=Q0 THEN POSITION 31,Q0:? MXL;"
Free";:TF=Q1:FIRST=Q1
680 POKE 53248,X*4+44
690 IF Y=0Y THEN 740
710 ADJOY=0Y*4+16:ADJY=Y*4+16
720 A=USR(CURSOR,PMB+ADJOY,Q0):A=USR(CURSOR,
PMB+ADJY,15):0Y=Y
740 K=PEEK(53279):IF K<7 THEN 2570
770 T=PEEK(764):IF T=255 OR T=39 OR T=154 TH
EN 740
775 IF T=166 THEN POKE 764,255:GOTO 1570
790 POKE 694,Q0:A=USR(GCHAR)
800 IF TF THEN 650
810 IF A<32 OR A>122 OR A=96 THEN 880
820 A=A-32*(A<96)
830 POKE SCR+X+L*Y,A
840 LOC=(CL+Y-Q1)*L2+X-Q1
850 TX$(LOC,LOC)=CHR$(A)
860 X=X+Q1-BF:GOSUB 460
870 BF=Q0:GOTO 680
880 IF A<>155 THEN 910
890 GOSUB 2640:POKE SCR+X+L*Y,94:TX$(LOC,LOC
+L2-X+Q1)=B$:X=2:Y=Y+1
900 TX$(LOC,LOC)=CHR$(94):GOSUB 460:GOTO 650
910 IF A=6 THEN A=127:GOTO 830
920 IF A=28 THEN Y=Y-Q1:GOSUB 460:GOTO 680
930 IF A=29 THEN Y=Y+Q1:GOSUB 460:GOTO 680
940 IF A=30 THEN X=X-Q1:GOSUB 460:GOTO 680
950 IF A=96 THEN A=74:GOTO 830
960 IF A=31 THEN X=X+Q1:GOSUB 460:GOTO 680
970 IF A=Q0 THEN A=72:GOTO 830
980 IF A=126 THEN X=X-Q1:GOSUB 460:A=Q0:BF=Q
1:GOTO 830
1040 IF A<>255 THEN 1070
1050 A=USR(EDCOM,ADR(TX$((CL+Y-Q1)*L2+X-Q1))
,ADR(TX$(MXL*L2+37)),Q0)
1060 GOTO 650
1070 IF A<>254 THEN 1100
1080 A=USR(EDCOM,ADR(TX$((CL+Y-Q1)*L2+X-Q1))
,ADR(TX$(MXL*L2+37)),Q1)
1090 GOTO 650
1100 IF A<>157 THEN 1160
1110 GOSUB 590:? "Insert Line";
1120 GOSUB 540:POKE 559,Q0

```

```

1130 FOR I=EOT+(EOT<MXL) TO CL+Y STEP -Q1:T$
    =TX$((I-Q1)*L2+Q1,I*L2):TX$(I*L2+Q1,I*L
    2+L2)=T$:NEXT I
1140 T=(CL+Y-Q1)*L2:TX$(T+Q1,T+L2)=B$
1150 X=2:POKE 559,46:GOTO 650
1160 IF A=159 THEN GOSUB 590:? "Tab set at "
    ;X-Q1:TF=Q1:TB$(X-Q1,X-Q1)="*":GOTO 740
1170 IF A=158 THEN GOSUB 590:? "Tab cleared
    at ";X-Q1:TF=Q1:TB$(X-Q1,X-Q1)=CHR$(Q0)
    :GOTO 740
1180 IF A<>127 THEN 1230
1190 IF TB$=B$ THEN GOSUB 590:? "No tabs set
    ":TF=Q1:GOTO 740
1200 FOR I=X TO L2:IF TB$(I,I)=CHR$(Q0) THEN
    NEXT I:T=L2:X=2:Y=Y+Q1:GOSUB 460:GOTO
    1200
1210 T=I:I=L2:NEXT I
1220 X=T+Q1:GOTO 680
1230 IF A<>156 THEN 1290
1240 GOSUB 590:? "Delete Line";
1250 GOSUB 540:POKE 559,Q0
1260 FOR I=CL+Y-Q1 TO EOT:T$=TX$((I+Q1)*L2+Q
    1,(I+2)*L2):TX$(I*L2+Q1,I*L2+L2)=T$:NEX
    T I
1270 T=EOT*L2:TX$(T+Q1,T+L2)=B$
1280 X=2:POKE 559,46:GOTO 650
1290 IF A=11 THEN GOSUB 590:TF=Q1:? "Clear a
    11 tabs":TB$=B$:GOTO 740
1320 IF A<>125 THEN 1450
1330 GOSUB 590:? "Erase: ";
1340 GOSUB 2650
1350 IF A=155 THEN 650
1355 IF A<>65 THEN 1370
1360 ? "ALL - ";:GOSUB 2540
1365 GOTO 613
1370 IF A<>82 THEN 1380
1372 ? "Remainder - ";:GOSUB 2540:GOSUB 2640
1375 TX$(LOC)=CHR$(Q0):TX$((MXL+Q23)*L2)=CHR
    $(Q0):TX$(LOC+Q1)=TX$(LOC):GOTO 650
1380 IF A<>68 OR NL=-Q1 THEN 650
1400 ? "Defined Lines - ";
1410 GOSUB 2540:POKE 559,Q0:GOSUB 540
1420 FOR I=FRL-Q1 TO EOT:T$=TX$((I+NL+Q1)*L2
    +Q1,(I+NL+2)*L2):TX$(I*L2+Q1,I*L2+L2)=T
    $:NEXT I
1430 FOR I=EOT-NL TO EOT:TX$(I*L2+Q1,I*L2+L2
    )=B$:NEXT I:NL=-Q1
1440 POKE 559,46:GOTO 650
1450 IF A<>4 THEN 1810

```

```

1460 GOSUB 590: ? "Define Lines";
1470 FL=CL:FR=Y:FRL=FL+FR:NL=Q0
1480 POKE SCR+1+L*(FR+NL),223
1490 LOC=CL*L2+(FR+NL-Q1)*L2:T=RL*L2+NL*L2:T
    $=TX$(LOC+Q1,LOC+L2):TX$(T+Q1,T+L2)=T$
1500 GOSUB 2650
1510 IF A=29 AND FR+NL<22 THEN NL=NL+Q1:GOTO
    1480
1520 IF A=28 AND FR+NL>FR THEN POKE SCR+1+L*
    (FR+NL),Q0:NL=NL-Q1
1530 IF A=155 THEN 1550
1540 GOTO 1500
1550 FOR I=Q0 TO NL:POKE SCR+1+L*(FR+I),Q0:N
    EXT I:GOTO 650
1570 POKE 53248,Q0:PRINT CHR$(125):POSITION
    13,Q0: ? " HELP Screen"
1580 ? "{DOWN}{TAB}{3 SPACES}Control Keys:"
1590 ? "[E]=Advance Page [E]=Page Back"
1595 ? "[D]=Define Lines [F]=Print format char.
    "
1610 ? "[K]=Kill all tabs [L]=Line Duplicate"
1620 ? "[P]=Print text{4 SPACES}[Q]=Quit"
1630 ? "Atari Key=Cancel Command":?
1635 ? "^x Print special character"
1640 ? "{DOWN}[C]LEAR[ ] Erase:[F]ll [D]efined Lin
    es":POKE 85,16: ? "[R]emainder"
1650 ? "[O]PTION[ ] Non-destructive CR"
1660 ? "{DOWN}[S]ELECT[ ] Filer:[R]ecall or [S]ave"
1670 ? "{DOWN}[I]START[ ] 'Home' cursor. Press
    twice to go to start of text."
1680 ? "{DOWN}[E]SC[ ] Mini DOS"
1700 ? "{DOWN}Press RETURN. ":A=USR(GCHAR):GO
    TO 650
1810 IF A<>12 THEN 1910
1820 GOSUB 590: ? "Duplicate defined lines";
1830 IF NL<Q0 THEN 650
1840 FOR I=Q0 TO NL
1850 IF CL+Y+I-Q1>MXL THEN I=NL:GOTO 1900
1860 T=RL*L2+I*L2
1870 T2=CL*L2+(Y+I-Q1)*L2
1880 T$=TX$(T+Q1,T+L2)
1890 TX$(T2+Q1,T2+L2)=T$
1900 NEXT I:Y=Y+NL+Q1:GOSUB 460:GOTO 650
1910 IF A<>27 THEN 2400
1920 POSITION 2,Q0: ? "[D]irectory,[L]ock,[U]nlock,
    [R]ename,[S]cratch?"
1930 GOSUB 2650:J=A
1940 IF J<>76 AND J<>85 AND J<>83 AND J<>68
    AND J<>82 THEN 1930

```

```

1950 IF J<>ASC("D") THEN 2020
1960 ? CHR$(125):POKE 53248,00
1970 TRAP 2170:OPEN #2,6,00,"D:*.*)"
1980 INPUT #2,T$:? T$:IF LEN(T$)<17 THEN 2000
1990 GOTO 1980
2000 CLOSE #2:TRAP 40000:GOSUB 590:?"Press
a key..";:OK=1:GOSUB 2650:IF A=27 THEN
1920
2010 GOTO 650
2020 GOSUB 590:J=A
2030 IF J=76 THEN ? " LOCK ";:J=35
2040 IF J=83 THEN ? " SCRATCH ";:J=33
2050 IF J=85 THEN ? " UNLOCK ";:J=36
2060 IF J=ASC("R") THEN 2130
2070 ? "Enter Filename:";
2080 MX=12:AL=01:GOSUB 2720
2090 T$(3)=IN$:T$(1,2)="D:":POSITION 10,00:?"
DEL$(1,15);
2100 TRAP 2170:IF J=33 THEN POSITION 24,00:G
OSUB 2540:COLOR 32:PLOT 24,00:DRAWTO 38
,00
2110 TRAP 2170:XIO J,#2,00,00,T$:TRAP 40000
2120 TRAP 40000:GOTO 650
2130 GOSUB 590:?" RENAME>Current name? ";:MX
=12:GOSUB 2720:T$(3)=IN$:T$(1,2)="D:"
2140 GOSUB 590:?" RENAME>New name? ";:MX=12:
GOSUB 2720:T$(LEN(T$)+01)="," :T$(LEN(T$
)+01)=IN$
2150 TRAP 2170:XIO 32,#2,00,00,T$:TRAP 40000
2160 GOTO 650
2170 TRAP 2170:POKE 559,46:CLOSE #2:CLOSE #3
:GOSUB 590:?"CHR$(253); "I/O Error #";PE
EK(195);:TF=01:GOTO 740
2180 GOSUB 590:?" Save or Recall";
2190 ICCOM=834+48:ICBAL=ICCOM+2:ICBLL=ICBAL+
4:ICSTAT=835+48:REM IOCB#3
2200 GOSUB 2650:IF A=155 THEN 1380
2210 IF A<>ASC("S") THEN 2290
2220 GOSUB 600:?"SAVE:{3 SPACES}File name?
";:MX=12:GOSUB 2720:T$(3)=IN$:T$(1,2)="
D:":GOSUB 550
2230 POSITION 5,0:?"DEL$(1,12); "ING";
2232 TRAP 2238:OPEN #3,4,00,T$:CLOSE #3:GOSU
B 600:?" REPLACE ";:IN$;" - ";:GOSUB 2
540
2233 GOSUB 600:?"REPLACING ";:IN$:GOTO 2240
2238 CLOSE #3:IF PEEK(195)<>170 THEN 2170
2240 TRAP 2170:OPEN #3,8,00,T$

```

```

2250 POKE ICCOM,11:P=ADR(TX$):POKE ICBAL+Q1,
    INT(P/256):POKE ICBAL,P-(INT(P/256)*256
    )
2260 LN=(CL+EOT+Q1)*L2:POKE ICBLL+Q1,INT(LN/
    256):POKE ICBLL,LN-(INT(LN/256)*256)
2270 A=USR(ADR(CIO$),48):ERR=PEEK(ICSTAT):PO
    KE 195,ERR:IF ERR>1 THEN 2170
2280 CLOSE #3:TRAP 40000:POKE 53279,Q0:GOTO
    650
2290 IF A<>ASC("R") THEN 650
2300 LK=Q0
2310 GOSUB 590:? "RECALL: Filename? ";;MX=12
    :GOSUB 2720:T$(3)=IN$:T$(1,2)="D:"
2315 LOC=(CL+Y-Q1)*L2+Q1:TX$(LOC)=CHR$(Q0):T
    X$((MXL+Q23)*L2)=CHR$(Q0):TX$(LOC+Q1)=T
    X$(LOC)
2320 TRAP 2170:POSITION 8,0:? DEL$(1,8);"ING
    ";;OPEN #3,4,Q0,T$
2330 ICCOM=834+48:ICBAL=ICCOM+2:ICBLL=ICBAL+
    4
2340 POKE ICCOM,5:P=ADR(TX$((CL+Y-Q1)*L2+Q1)
    ):POKE ICBAL+Q1,INT(P/256):POKE ICBAL,P
    -(INT(P/256)*256)
2350 LN=(MXL-(CL+Y-Q1))*L2:POKE ICBLL+Q1,INT
    (LN/256):POKE ICBLL,LN-(INT(LN/256)*256
    )
2360 A=USR(ADR(CIO$),48):ERR=PEEK(ICSTAT):PO
    KE 195,ERR:IF ERR>1 AND ERR<>136 THEN 2
    170
2370 CLOSE #3:POKE 53279,Q0:TRAP 40000:IF LK
    =Q0 THEN 650
2380 CL=Q0:Y=Q1:X=2:T=USR(SCRZAP,ADR(TX$))
2390 GOTO 2950
2400 IF A<>17 THEN 2410
2403 GOSUB 600:? "Quit: ";;GOSUB 2540
2405 POKE 53277,Q0:POKE 53248,Q0:POKE 53774,
    192:POKE 16,192:GRAPHICS Q0:POKE 702,64
    :END
2410 IF A=16 THEN 2840
2420 IF A=Q1 THEN CL=CL+Q23:Y=Q1:GOSUB 460:G
    OTO 650
2430 IF A=2 THEN CL=CL-Q23:Y=Q1:GOSUB 460:GO
    TO 650
2500 GOTO 640
2540 ? "Are you sure?";:GOSUB 2650:IF 1-(A=1
    21 OR A=89) THEN POP :GOTO 650
2550 RETURN
2570 REM *** Handle console keys
2580 POKE 764,130:A=USR(GCHAR):POKE 77,Q0

```



```

2590 IF K=5 THEN 2180
2600 IF K=3 THEN X=2:Y=Y+Q1:GOSUB 460:GOTO 6
80
2610 IF K=6 AND Y=Q1 AND X=2 THEN CL=Q0:X=2:
GOTO 650
2620 IF K=6 THEN Y=Q1:X=2:GOTO 650
2630 GOTO 740
2640 LOC=(CL+Y-Q1)*L2+X-Q1:RETURN
2650 T=Q0:REM GET A KEY
2660 IF PEEK(20)>20 THEN T=Q1-T:POKE 20,Q0:P
OKE 755,T*2
2665 IF OK THEN IF PEEK(53279)=5 THEN POKE 7
55,2:POKE 559,46:POP:POKE 764,130:A=US
R(GCHAR):OK=0:GOTO 2180
2670 IF PEEK(764)=255 THEN 2660
2680 IF PEEK(764)=154 THEN 2660
2690 IF PEEK(764)=39 THEN POKE 764,255:SOUND
Q0,5,12,4:POP:FOR T=1 TO 5:NEXT T:SOU
ND Q0,Q0,Q0,Q0:GOSUB 2710:GOTO 650
2700 TRAP 2700:A=USR(GCHAR):TRAP 40000:IF A>
96 AND A<123 THEN A=A-32*AL
2710 POKE 755,2:POKE 559,46:RETURN
2720 REM *** PSEUDO-INPUT
2730 IN$=""
2740 ? CHR$(21);CHR$(30);:GOSUB 2650: ? CHR$(
32);CHR$(30);
2750 IF A=155 THEN 2820
2760 IF A=126 AND LEN(IN$)>1 THEN IN$=IN$(1,
LEN(IN$)-Q1): ? CHR$(A);:GOTO 2740
2770 IF A=126 AND LEN(IN$)=Q1 THEN ? CHR$(A)
;:GOTO 2730
2780 IF LEN(IN$)=MX THEN 2740
2790 IF (A<32 OR A>90) AND A<96 OR A>122 THE
N 2740
2800 ? CHR$(A);:IN$(LEN(IN$)+Q1)=CHR$(A)
2810 GOTO 2740
2820 AL=Q1:IF IN$="" THEN POP:GOTO 650
2830 RETURN
2840 REM *** Printer Output
2850 GOSUB 590: ? "PRINT: (C/F)"
2860 CON=Q0:F=Q0:FOR I=Q0 TO 9:PC(I)=48+I:NE
XT I
2870 GOSUB 2650:IF A=155 THEN 2910
2880 IF A=67 THEN CON=1-CON:POSITION 10,Q0: ?
CHR$(67+128*CON);:GOTO 2870
2890 IF A=70 THEN F=1-F:POSITION 12,Q0: ? CHR
$(70+128*F):GOTO 2870
2900 GOTO 2870
2910 TRAP 2170:OPEN #2,8,Q0,"P:"

```

```

2920 GOSUB 590: ? "Printing..."
2930 LM=5:RM=75:CN=00:NL=00
2940 SP=1:PL=66:LP=50:C=LM
2950 GOSUB 540: IF F=1 THEN POKE 559,00
2960 FOR P=Q1 TO LC
2970 IF PEEK(764)<255 THEN GOSUB 2650:POP :G
OTO 3140
2980 Z=ASC(TX$(P))
2990 IF CN=Q1 AND Z<>127 THEN 3460
3000 IF Z<62 OR (Z>96 AND Z<123) THEN 3070
3010 IF Z=94 THEN GOSUB 3210:GOSUB 3150:GOTO
3120
3020 IF Z=72 THEN UL=Q1:PUT #2,27:PUT #2,52:
GOTO 3120
3030 IF Z=74 THEN UL=Q0:PUT #2,27:PUT #2,53:
GOTO 3120
3040 T=ASC(TX$(P+Q1)):IF Z=62 AND T>15 AND T
<26 THEN PUT #2,PC(T-16):P=P+1:GOTO 312
0
3060 IF Z=127 THEN 3230
3070 IF C=LM THEN FOR I=Q1 TO LM:PUT #2,32:N
EXT I
3080 C=C+1
3090 PUT #2,Z+32*(Z<64)
3100 T=Q0:IF RM-C>=10 THEN 3110
3105 FOR I=1 TO LEN(BRK$):IF Z+32<>ASC(BRK$(
I,I)) THEN NEXT I:GOTO 3110
3107 TT=ASC(TX$(P+Q1)):IF TT=Q0 OR TT=94 OR
Z=Q0 OR Z=13 THEN I=LEN(BRK$):NEXT I:GO
SUB 3150:T=Q1
3110 IF T=Q1 AND ASC(TX$(P+Q1))=Q0 THEN P=P+
Q1:IF P<LC THEN 3110
3120 NEXT P
3130 GOSUB 3150
3140 PRINT #2:CLOSE #2:POKE 559,46:TRAP 4000
0:GOTO 650
3150 FOR I=Q1 TO SP:PRINT #2:NEXT I
3160 C=LM:NL=NL+SP:IF CN<Q0 THEN CN=Q1
3170 IF NL<LP THEN RETURN
3180 IF CN=Q0 THEN FOR I=Q0 TO 255 STEP 17:
SOUND Q0,255-I,10,15-INT(I/17):NEXT I:T
=USR(6CHAR):GOTO 3200
3190 FOR I=Q1 TO PL-LP:PRINT #2:NEXT I
3200 NL=Q0:RETURN
3210 REM *** SKIP TRAILING BLANKS
3220 T=INT(P/L2):P=(T+Q1-(P/L2=T))*L2:RETURN

3230 REM Handle special formatting
3240 P=P+Q1

```

```

3250 CM$=TX$(P,P+Q1):T$=""
3260 FOR I=P+2 TO LC
3270 IF TX$(I,I)>=CHR$(16) AND TX$(I,I)<CHR$(
    26) THEN T$(LEN(T$)+Q1)=CHR$(ASC(TX$(I
    ,I))+32):NEXT I
3280 V=Q0:P=I:TRAP 3290:V=VAL(T$)
3290 TRAP 2170:IF CM$="cn" THEN CN=V
3300 IF CM$="ln" THEN FOR J=Q1 TO V:GOSUB 31
    50:NEXT J
3310 IF CM$="sp" THEN SP=V
3320 IF CM$="pl" THEN PL=V
3330 IF CM$="lp" THEN LP=V
3340 IF CM$="lm" AND V>0 THEN LM=V:C=V
3350 IF CM$="rm" AND V>0 THEN RM=V
3360 IF CM$="fp" THEN GOSUB 3180:POKE 559,46
    -46*F
3370 IF CM$="as" THEN PUT #2,V
3380 IF CM$="cm" THEN FOR I=P TO P+79:IF TX$(
    I,I)<>"^" THEN NEXT I:I=I-Q1
3390 IF CM$="cm" THEN P=I+Q1:GOTO 3450
3400 IF CM$<>"nf" THEN 3430
3410 T$="D:":FOR I=Q0 TO 11:Z=ASC(TX$(P+I,P+
    I)):IF Z<>94 AND P+I<=LC THEN T$(3+I)=C
    HR$(Z+32*(Z<63)):NEXT I
3415 TX$(Q1)=CHR$(Q0):TX$((MXL+Q23)*L2)=CHR$(
    Q0):TX$(2)=TX$
3420 POKE 559,46:GOSUB 590:?"Printing ";T$:
    LK=Q1:CL=Q0:Y=Q1:GOTO 2320
3430 IF ASC(CM$)>15 AND ASC(CM$)<26 THEN PC(
    ASC(CM$)-16)=V
3440 IF TX$(P,P)<>"^" AND P<LC THEN 3240
3450 GOSUB 3220:P=P+Q1:GOTO 2970
3460 REM *** CENTER STRING
3470 LN=Q0:FOR I=P TO P+79:IF TX$(I,I)<>"^"
    THEN LN=LN+Q1:NEXT I
3480 WIDTH=RM-LM:UL=Q0:IF TX$(P,P)=CHR$(72)
    THEN UL=Q1
3490 FOR I=Q1 TO (WIDTH-LN)/2+LM:PUT #2,32:N
    EXT I
3500 C=C+I:CN=-Q1:GOTO 2990
5000 REM INITIALIZATION
5010 GRAPHICS 17:SETCOLOR 4,1,10
5020 DL=PEEK(560)+256*PEEK(561)+4:POKE DL+5,
    7:POKE DL+10,7:POKE DL+14,7
5030 POSITION 6,4:?" #6;"Scriptor":POSITION 3
    ,7:?" #6;"WORD PROCESSOR"
5040 ?" #6:?" #6;" ";CHR$(136);CHR$(227);CHR$(
    137);" Copyright";CHR$(145);CHR$(153);
    CHR$(152);CHR$(147)

```

```

5045 ? #6: ? #6; "{4 SPACES}compute{A} publ";C
HR$(14);
5050 ? #6: ? #6; "{3 SPACES}CHARLES BRANNON"
5070 Q0=0: Q1=1: Q23=23: RL=MXL+Q1: SCRZAP=1680:
CURSOR=1739: EDCOM=1536: AL=1: L2=38: GCHAR
=1303: SND=1331
5080 DIM T$(79), IN$(20), B$(L2), TB$(L2), CM$(2
), BRK$(8), PC(9), DEL$(20), CIO$(7)
5090 B$=CHR$(Q0): B$(L2)=B$: B$(2)=B$: DEL$=CHR
$(254): DEL$(20)=DEL$: DEL$(2)=DEL$
5100 TB$=B$: BRK$=" , . ! ? ; : - " : CIO$="hhh": CIO$(
4)=CHR$(170): CIO$(5)="LV": CIO$(7)=CHR$(
228)
5110 OPEN #1,4,Q0,"K:"
5120 T=Q0: OY=Q0: CL=Q0: L=40: NL=-Q1
5130 PMB=PEEK(106)-8: POKE 559,46: POKE 53248,
Q0
5140 POKE 54279,PMB: POKE 53277,3
5150 PMB=PMB*256+512: POKE 704,56
5160 FOR I=Q0 TO 255: POKE PMB+I,Q0: POKE 708+
3*RND(Q0),PEEK(53770): NEXT I
5180 SETCOLOR 4,8,2
5250 FOR I=0 TO 70: READ A: POKE 1280+I,A: CHEC
KSUM=CHECKSUM+A: POKE 708+3*RND(Q0),PEEK
(53770): NEXT I
5290 FOR I=0 TO 247: READ A: POKE 1536+I,A: CHE
CKSUM=CHECKSUM+A: POKE 708+3*RND(Q0),PEE
K(53770): NEXT I
5300 IF CHECKSUM<>47765 THEN PRINT CHR$(253)
;"Error in DATA statements...":END
5310 DATA 72,138,72,169,10,162,2,141,10,212,
141,24,208,141,26,208,142,23,208,104,17
0,104,64
5320 DATA 104,173,252,2,201,255,240,249,133,
124,162,255,142,252,2,32,51,5,32,254,24
6,133,212,169,0,133,213,96
5330 DATA 162,0,142,0,210,162,15,142,1,210,1
60,0,234,200,208,252,202,16,244,96
5340 DATA 216,104,104,133,213,104
5350 DATA 133,212,104,133,204,104
5360 DATA 133,203,104,104,208,47
5370 DATA 32,109,6,165,205,76
5380 DATA 43,6,160,0,177,205
5390 DATA 200,145,205,198,205,165
5400 DATA 205,201,255,208,2,198
5410 DATA 206,197,212,208,235,165
5420 DATA 206,197,213,208,229,160
5430 DATA 0,177,205,200,145,205
5440 DATA 136,152,145,205,96,201

```

```

5450 DATA 1,240,3,76,221,6
5460 DATA 32,109,6,76,91,6
5470 DATA 160,1,177,212,136,145
5480 DATA 212,230,212,208,2,230
5490 DATA 213,165,213,197,206,208
5500 DATA 237,165,212,197,205,208
5510 DATA 231,169,0,168,145,212
5520 DATA 96,165,212,133,205,165
5530 DATA 213,133,206,160,0,177
5540 DATA 205,201,94,240,18,230
5550 DATA 205,208,2,230,206,165
5560 DATA 206,197,204,208,238,165
5570 DATA 205,197,203,208,232,96
5580 DATA 165,88,133,203,165,89
5590 DATA 133,204,104,104,133,206
5600 DATA 104,133,205,162,24,76
5610 DATA 188,6,160,0,177,205
5620 DATA 200,200,145,203,136,192
5630 DATA 38,208,245,24,169,38
5640 DATA 101,205,133,205,144,2
5650 DATA 230,206,24,169,40,101
5660 DATA 203,133,203,144,2,230
5670 DATA 204,202,208,218,96,104
5680 DATA 104,133,204,104,133,203
5690 DATA 104,168,104,145,203,200
5700 DATA 192,4,208,249,96,160
5710 DATA 0,177,212,208,20,198
5720 DATA 212,165,212,201,255,203
5730 DATA 2,198,213,197,203,208
5740 DATA 238,165,213,197,204,208
5750 DATA 232,96
6000 GRAPHICS 0:POKE 559,00:POKE 16,64:POKE
53774,64
6010 FOR I=5000 TO 5900 STEP 100: ? CHR$(125)
: POSITION 2,3:FOR J=I+90 TO I STEP -10:
? J:NEXT J: ? 110: ? "CONT"
6020 POKE 712,PEEK(53770):POKE 842,13:POSITI
ON 0,0:STOP
6030 POKE 842,12:NEXT I
6040 SETCOLOR 2,12,00:SETCOLOR 4,8,10:SETCOL
OR 01,00,12:POKE 752,01
6050 POKE PEEK(560)+256*PEEK(561)+3,194:POKE
512,0:POKE 513,5
6060 ? CHR$(125): ? : ? :FOR I=6000 TO 6060 ST
EP 10: ? I:NEXT I: ? "GOTO610":POSITION 0
,0:POKE 842,13:STOP

```

Program 2. Scriptor Modification for 825 Printer

Change these lines in Program 1 if you have an 825 printer.

```
3020 IF Z=72 THEN UL=01:PUT #2,15:GOTO 3120
3030 IF Z=74 THEN UL=00:PUT #2,14:GOTO 3120
3070 IF C=LM THEN PUT #2,14:FOR I=01 TO LM:P
    UT #2,32:NEXT I:PUT #2,15*UL
```

Program 3. Scriptor for 1200XL

Change these lines in Program 1 if you have an Atari 1200XL.

```
750 IF PEEK(732) THEN POKE 732,0:GOTO 1570
5300 IF CHECKSUM<>47596 THEN PRINT CHR$(253)
    ;"Error in DATA statements...":END
5320 DATA 104,173,252,2,201,255,240,249,133,
    124,162,255,142,252,2,32,51,5,32,89,242
    ,133,212,169,0,133,213,96
6000 GRAPHICS 0:POKE 559,00:POKE 16,64:POKE
    53774,64:POKE 731,255
7000 REM
```

Program 4. Scriptor for 800XL

Change these lines in Program 1 if you have an Atari 800XL.

```
5300 IF CHECKSUM<>47543 THEN PRINT CHR$(253)
    ;"Error in DATA statements...":END
5320 DATA 104,173,252,2,201,255,240,249,133,
    124,162,255,142,252,2,32,51,5,32,35,243
    ,133,212,169,0,133,213,96
```

THE UNIVERSITY OF CHICAGO
DEPARTMENT OF THE HISTORY OF ARTS
AND ARCHITECTURE
1100 EAST 58TH STREET
CHICAGO, ILLINOIS 60637

RECEIVED
JAN 10 1964
FROM THE
LIBRARY OF THE
UNIVERSITY OF CHICAGO
DEPARTMENT OF THE HISTORY OF ARTS
AND ARCHITECTURE
1100 EAST 58TH STREET
CHICAGO, ILLINOIS 60637

THE UNIVERSITY OF CHICAGO
DEPARTMENT OF THE HISTORY OF ARTS
AND ARCHITECTURE
1100 EAST 58TH STREET
CHICAGO, ILLINOIS 60637

4 Graphics

4

SuperFont Plus

John Slaby and Charles Brannon

You can generate excellent game graphics by using ANTIC modes 4 and 5. This program provides an ANTIC version of SuperFont. Requires 16K RAM.

After typing in "SuperFont" (*COMPUTE!'s First Book of Atari Graphics*), I was very pleased. I couldn't imagine needing any additional functions or purchasing any font editor that could possibly improve upon it. Then I bought *De Re Atari*, and everything I had read previously in the *Hardware Manual* on ANTIC modes 4 and 5 fell into place. At the same time I realized that it was ANTIC mode 4 that allowed the great graphics in *Caverns of Mars*. I realized I *could* make some useful additions to the original program. Therefore, I offer SuperFont Plus.

Charles Brannon stated in his article on SuperFont that it would be easy to expand the program, so I did. The additional commands are the ANTIC, PRINT, DATA, and Color Change. Of these, only the DATA and PRINT commands can be used along with the original version of Graphics modes 0, 1, and 2. This expanded version is about 65 percent longer and, if you have only 16K RAM, some manipulation will be required; but you can have an ANTIC version of SuperFont. For those of you that already have SuperFont, just add lines 10, 20, 105, 106, 115, 375, 475, 477, 1415, and 1601 through 1605. Also note the changes in lines 100, 120, 190, 270, 320, 340, 380, 470, 480, 590, 650, 1300, 1320, 1360, 1370, 1400, 1410, and 1420. If you obtained your SuperFont from *COMPUTE!'s First Book of Atari Graphics*, you will also have to delete line numbers 5000 on up, as there is no room in the menu for the printer command. Once you do this, you will have the capability of designing your own ANTIC 4/5 character set.

For those of you with only 16K, there is a way out. You will have to end up with two fonts: one font, the original, for the BASIC-supported graphics modes, and one for the ANTIC 4/5 graphics modes. If you delete the following commands and change lines 250 and 300 to say RAM-4 instead of RAM-8, you will have a functional font. The deleted commands which have limited use for ANTIC 4/5 are: RESTORE (920-930), OVERLAY

(870-910), GRAPHICS (1370-1390), WRITE DATA (1290-1360), and QUIT (1130-1140). Also, please do not add the DATA command; you will not have enough memory to use it. I've included a utility that will read the saved character set from S command and put the character set into DATA lines just as the full-fledged version of SuperFont Plus does. Be sure to change 3500 to 520 in line 3000 so you don't jump to the DATA command that doesn't exist.

Original SuperFont

Here's a quick review of the original SuperFont commands:

EDIT: The character you select by pressing the joystick trigger is copied to the grid in the upper section of the screen. The cursor is relocated to this grid, and you can instantly modify the character by moving the joystick and pressing the trigger to either set or remove a point, as desired.

RESTORE: This will copy the pattern from the first character set to the second, located in the lower half of the screen.

COPY FROM: Selects a character which will be copied to the current one you are working on.

COPY TO: The current character will be copied to the selected place.

SWITCH: Exchanges the current character for the one selected.

OVERLAY: Adds the selected character's pattern to the current one.

CLEAR: Clears the pattern of the current character. A must for ANTIC 4/5.

INVERT: Turns current character upside down.

SAVE FONT: Saves character set to disk or tape. Answer "Filename" with either C: or D:filespec. If you see an error message, press any key to return to the menu.

LOAD FONT: Retrieves a character set that you saved. Answer "Filename" here the same way as in SAVE FONT.

CURSOR-UP or SHIFT DELETE: The line of points the cursor is on is deleted, and the following lines are pulled up to fill the gap.

CURSOR-DOWN or SHIFT INSERT: A blank line is inserted on the line the cursor is in, and all lines below it move down one. The bottom line is lost.

SCROLL LEFT: The bit pattern of the character is shifted left.

SCROLL RIGHT: The bit pattern of the character is shifted right.

WRITE DATA: The internal code (0-127) of the character and the eight bytes that make it up are displayed in the menu area. Press any key to return to menu.

GRAPHICS: This toggles the TEXT/GRAPHICS option of graphics modes 1 and 2 to let you see each half of the character set.

REVERSE: All blanks become points, and vice versa. Works the same as pressing the Atari logo key and then typing.

QUIT: Exit program.

SuperFont Plus: Three New Commands

The ANTIC(A) command mode modifies the display list so that the lower section of the screen now becomes ANTIC mode 4 except for the last line, which is ANTIC 5. Press A again to return to the original GRAPHICS 0, 1, and 2. Once you activate this command, the character set will become mostly unrecognizable. This is because the characters are now four pixels wide instead of eight, but the overall displayed width remains the same. This loss of resolution is the price you have to pay for the multicolor ability of these ANTIC modes.

Use all other commands as before; they will work. Please note that the grid now has double-wide pixels when compared to the first display. This is because that binary number you place in each pixel determines the color that will be displayed and you need two bits per color. The binary number is related to the color registers as follows: 00 = Background; 01 = Playfield 0; 10 = Playfield 1; and 11 = Playfield 2. To use Playfield 3's color, you also use binary 11, but the internal code must be 128-255. This is accomplished by using reversed characters via the Atari logo key. There is no way to use this key in any of the original commands, so the PRINT command was created.

The PRINT mode (P) allows you to print any character in the bottom window next to another one just as in normal typing. This mode allows you to see that third playfield color via the logo key. You can type as long as you like, but if you exceed 38 characters, the first one will be lost and all the others will shift left. As noted before, this command can be used with the original GRAPHICS 1 and 2.

Since the keyboard is used for typing, the START and SELECT buttons will, respectively, return you to the menu and clear the typing area. When you return to the menu, the typing

area isn't automatically cleared; this allows you to work on more than one character at a time, that is, three characters together as a car, etc. This mode is also useful to get a full screen effect for one line of modified characters.

The next new command is the Color Change mode (K). When I started working with the first two new commands, it became obvious that the ability to change the color of the character I was working on would be very useful. Thus I expanded the Display List Interrupt to give me that ability and added a second interrupt for the background color change.

When you activate this command, you will be able to change only the colors for the ANTIC 4/5 character set. If you want to change the colors for the original graphics modes, modify lines 170 and 300 as desired. The menu area will be cleared, and you will be given the choice of the playfield or background color you want to change. If you change the background, it will affect only the typing window. I did this to keep the clarity of the character set at its best, and you will probably want to see the change for only one or two characters at a time.

After your register selection, you will be asked for the color and luminosity value (0-14) you want. To help you, a list of colors will be supplied in the menu area. If you give a bad input, you will be asked to try again, starting with the color value. To get the decimal value being used by that register, press R when being offered the color registers and then select a register.

Using the Character Set

Once you have created the character set, you will need to save it to disk or tape. There are a number of options open to you. The first one was supplied in the original SuperFont, the S (SAVE) font option. To use this, just press S and respond to the filename prompt with either C: for cassette or D: *filename* for disk.

There are two methods of using the character set saved with the S option. Program 2 (Character Set Loader) simply loads the set into memory and changes CHBASE(756) to point to it.

The second method is to use Program 3 (Character Set DATA-maker) to create a module of lines that lets you add your character set to any program. After saving your character set to tape or disk, just RUN Program 3. It will ask you for the filename of the character set, the starting line number of the module, and a filename for the module. (Just answer C: to the filename prompt for use with a cassette.)

Program 3 writes a subroutine replete with the appropriate PEEKs, loops, and DATA statements, to tape or disk. It optimizes space by writing DATA statements only for those characters you have changed. After it has finished, you can use the ENTER command to merge the statements produced with any program. You may need to make some minor adjustments to the DATA statements it produces. Also, in your main program, remember to use a POKE 756,CHSET/256 after every GRAPHICS statement, since a GRAPHICS statement resets the character pointer.

The second method for saving your font is the D DATA STM command. To use this command, just press D. You will first be asked if you want to delete any character set line numbers. Once the lines have been deleted, or if you do not wish to delete lines, the save font prompts will begin.

To save the font, you must supply the starting line number (no line number below 4000 will be accepted) and the beginning and ending character. Once these inputs are completed, lines will be added starting with whatever line number was entered, incrementing by ten for each character. You can now LIST the font DATA to a printer, cassette, or disk. This file can now be merged with a program.

The subroutine below can be used within a program to POKE the new character set into memory from the DATA statements. Remember to POKE 756,CHSET/256 after each GRAPHICS statement.

```

3000 CHSET=PEEK(106)-8
3010 CHSET=CHSET*256
3020 FOR I=0 TO 1023
3030 READ A:POKE CHSET+I,A
3040 NEXT I
3050 RETURN

```

That covers everything; now you should be able to generate some excellent graphics characters like those in *Caverns of Mars* and *Eastern Front*.

Program 1. Superfont Plus

```

10 GOTO 100
20 POKE 82,14:POSITION 14,0:FOR I=ST TO ED:?
  "{25 SPACES}":NEXT I:RETURN
100 REM *** SUPERFONT + ***
105 REM Character Set Editor
140 DIM I(7),FN$(14),N$(10)
150 IF PEEK(1536)<>72 THEN GOSUB 1400

```

```

160 GRAPHICS 0:POKE 752,1
170 SETCOLOR 2,7,2:SETCOLOR 4,7,2
180 DL=PEEK(560)+256*PEEK(561)+4
190 SD=PEEK(88)+256*PEEK(89)+12*40:ASD=SD+5*
  40
200 A1=1630:FUNC=1631:A2=1632:LOGIC=1628
210 RAM=PEEK(106)-8:PMBASE=RAM*256
220 CHRORG=57344
230 POKE 559,46:POKE 54279,RAM
240 POKE 53277,3:POKE 53256,3
250 CHSET=(RAM-8)*256
260 POKE DL+23,6:POKE DL+24,7
270 POKE DL+17,130:POKE DL+18,112
280 POKE 512,0:POKE 513,6
290 POKE 54286,192
300 POKE 1549,RAM-8:POKE 1672,RAM-8:POKE 153
  8,0
310 A=USR(1555,CHSET)
320 P0=PMBASE+512+20:P1=PMBASE+640+20:P2=PMB
  ASE+768+20:P=PMBASE+896+20:T=85:GOSUB 33
  0:GOTO 350
330 FOR I=0 TO 7:FOR J=0 TO 3:T=255-T:POKE P
  0+I*4+J,0:POKE P1+I*4+J,T:T=255-T
340 POKE P2+I*4+J,T:NEXT J:T=255-T:NEXT I:RE
  TURN
350 POKE 53248,64:POKE 53249,64:POKE 53250,6
  4
360 POKE 704,198:POKE 705,240:POKE 706,68
370 POKE 53256,3:POKE 53257,3:POKE 53258,3:P
  OKE 623,1
375 GOSUB 380:GOSUB 390:GOTO 490
380 POSITION 2,0:?" {Q}{8 R}{E}":FOR I=1 TO
  8:?" {8 SPACES}!":NEXT I:?" {Z}{8 R}
  {C}":RETURN
390 POKE 82,14:POSITION 14,0
400 ? "[F] Edit{8 SPACES}[F] Restore"
410 ? "[F] Copy From{3 SPACES}[F] Switch"
420 ? "[F] Copy To{5 SPACES}[F] Clear "
430 ? "[F] Overlay{5 SPACES}[F] Invert"
440 ? "[F] Save Font{3 SPACES}[F] Load Font"
450 ? "{ESC}{DEL LINE} Delete{6 SPACES}{ESC}
  {INS LINE} Insert"
460 ? "{ESC}{CLR TAB} Scroll Left {ESC}
  {SET TAB} Scroll RT"
470 ? "[F][F][F] Reverse{3 SPACES}[F] Graphics
  "
475 ? "[F] Print mode [F] ANTIC 4&5"
477 ? "[F] Color change[F] DATA STM"
480 ? "[F] Write Data [F] Quit":RETURN

```

```
490 FOR I=0 TO 3:FOR J=0 TO 31:POKE SD+J+I*4
  0+4,I*32+J:POKE ASD+J+I*40+4,I*32+J:NEXT
  J:NEXT I:?
510 OPEN #2,4,0,"K:"
520 P=PEEK(764):IF P=255 THEN 520
525 POKE 82,2:POSITION 0,0
530 IF P=60 THEN 520
540 IF P=39 THEN POKE 764,168
550 GET #2,K
560 IF K<>ASC("E") THEN 790
570 GOSUB 1750
580 FOR I=0 TO 7:A=PEEK(CHSET+C*8+I):FOR J=0
  TO 3:POKE P0+I*4+J,A:NEXT J:NEXT I
590 POKE ASD+169+(ANTIC*10),C:POKE ASD+190+(
  ANTIC*30),C
600 JX=0:JY=0
610 POSITION JX+4,JY+1
620 ? CHR$(32+128*FF);"{LEFT}";:FF=1-FF
630 IF STRIG(0)=0 THEN 750
640 IF PEEK(764)<255 THEN ? " ";:GOTO 520
650 ST=STICK(0):IF ST=15 THEN 610
660 IF STRIG(0) THEN FOR I=0 TO 100 STEP 20:
  SOUND 0,100-I,10,8:NEXT I
670 POSITION JX+4,JY+1:? " ";
680 JX=JX+(ST=7)-(ST=11)
690 JY=JY+(ST=13)-(ST=14)
700 IF JX<0 THEN JX=7
710 IF JX>7 THEN JX=0
720 IF JY<0 THEN JY=7
730 IF JY>7 THEN JY=0
740 GOTO 610
750 POKE A1,PEEK(CHSET+C*8+JY):POKE A2,2^(7-
  JX):POKE FUNC,73:A=USR(LOGIC)
760 POKE CHSET+C*8+JY,A:FOR J=0 TO 3:POKE P0
  +JY*4+J,A:NEXT J
770 FOR I=0 TO 10:SOUND 0,I*4,8,8:NEXT I:SOU
  ND 0,0,0,0
780 GOTO 650
790 IF K<>ASC("F") THEN 830
800 S=C:GOSUB 1750
810 FOR I=0 TO 7:A=PEEK(CHSET+C*8+I):POKE CH
  SET+S*8+I,A:NEXT I
820 C=S:GOTO 580
830 IF K<>ASC("T") THEN 870
840 S=C:GOSUB 1750
850 FOR I=0 TO 7:A=PEEK(CHSET+S*8+I):POKE CH
  SET+C*8+I,A:NEXT I
860 C=S:GOTO 600
870 IF K<>ASC("O") THEN 920
```



```

880 S=C:GOSUB 1750
890 FOR I=0 TO 7:POKE A1,PEEK(CHSET+C*8+I):P
   OKE A2,PEEK(CHSET+S*8+I):POKE FUNC,9:A=U
   SR(LOGIC)
900 POKE CHSET+S*8+I,A:NEXT I
910 C=S:GOTO 580
920 IF K<>ASC("R") THEN 940
930 FOR I=0 TO 7:POKE CHSET+C*8+I,PEEK(CHROR
   G+C*8+I):NEXT I:GOTO 580
940 IF K<>ASC("C") THEN 960
950 FOR I=0 TO 7:POKE CHSET+C*8+I,0:NEXT I:G
   UTO 580
960 IF K<>ASC("{R}") THEN 980
970 FOR I=0 TO 7:POKE CHSET+C*8+I,255-PEEK(C
   HSET+C*8+I):NEXT I:GOTO 580
980 IF K<>ASC("X") THEN 1010
990 S=C:GOSUB 1750
1000 FOR I=0 TO 7:A=PEEK(CHSET+S*8+I):POKE C
   HSET+S*8+I,PEEK(CHSET+C*8+I):POKE CHSET
   +C*8+I,A:NEXT I:GOTO 580
1010 IF K<>ASC("I") THEN 1030
1020 FOR I=0 TO 7:I(I)=PEEK(CHSET+C*8+I):NEX
   T I:FOR I=0 TO 7:POKE CHSET+C*8+I,I(7-I
   ):NEXT I:GOTO 580
1030 IF K<>ASC("{UP}") AND K<>ASC("
   {DEL LINE}") THEN 1050
1040 FOR I=JY TO 6:POKE CHSET+C*8+I,PEEK(CHS
   ET+C*8+I+1):NEXT I:POKE CHSET+C*8+7,0:G
   OTO 580
1050 IF K<>ASC("{DOWN}") AND K<>ASC("
   {INS LINE}") THEN 1070
1060 FOR I=7 TO JY STEP -1:POKE CHSET+C*8+I,
   PEEK(CHSET+C*8+I-1):NEXT I:POKE CHSET+C
   *8+JY,0:GOTO 580
1070 IF K<>ASC("{LEFT}") THEN 1100
1080 FOR I=0 TO 7:A=PEEK(CHSET+C*8+I)*2:IF A
   >255 THEN A=A-256
1090 POKE CHSET+C*8+I,A:NEXT I:GOTO 580
1100 IF K<>ASC("{RIGHT}") THEN 1130
1110 FOR I=0 TO 7:A=INT(PEEK(CHSET+C*8+I)/2)
1120 POKE CHSET+C*8+I,A:NEXT I:GOTO 580
1130 IF K<>ASC("Q") THEN 1150
1140 POKE 53248,0:POKE 53249,0:POKE 53250,0:
   POKE 53277,0:GRAPHICS 0:END
1150 IF K<>ASC("S") THEN 1210
1160 GOSUB 1610:POKE 195,0
1170 TRAP 1190:OPEN #1,8,0,FN$
1180 A=USR(1589,CHSET)
1190 CLOSE #1:TRAP 40000:IF PEEK(195) THEN 1
   260

```

```

1200 POKE 54286,192:GOSUB 390:GOTO 520
1210 IF K<>ASC("L") THEN 1290
1220 GOSUB 1610:POKE 195,0
1230 TRAP 1250:OPEN #1,4,0,FN$
1240 A=USR(1619,CHSET)
1250 CLOSE #1:TRAP 40000:IF PEEK(195)=0 THEN
    1200
1260 POSITION 14,0:? "{BELL}*ERROR -";PEEK(1
    95);"*"
1270 IF PEEK(764)<255 THEN POSITION 14,0:? "
    {19 SPACES}":GOTO 1200
1280 GOTO 1270
1290 IF K<>ASC("W") THEN 1370
1300 ST=0:ED=11:GOSUB 20:N$="{3 SPACES}":L=L
    EN(STR$(C)):N$(1,L)=STR$(C):L=LEN(N$):P
    OSITION 14,0
1310 FOR I=1 TO L:? CHR$(ASC(N$(I,I))+128);:
    NEXT I:? ">";
1320 Z=0:FOR I=0 TO 2:FOR J=0 TO 1+(I>0):A=P
    EEK(CHSET+C*8+Z):Z=Z+1
1330 SOUND 0,(I*3+J)*10+50,10,8
1340 ? A;",";NEXT J:NEXT I:SOUND 0,0,0,0
1350 IF PEEK(764)=255 THEN 1350
1360 GOSUB 20:GOSUB 390:GOTO 520
1370 IF K<>ASC("G") THEN 2000
1380 CF=1-CF:POKE 1549,8+2*CF
1390 GOTO 520
1400 GRAPHICS 2+16:SETCOLOR 4,1,4:POSITION 5
    ,3:? #6;"SUPERFONT +"
1410 POSITION 5,5:? #6;"patience{3 N}":POSIT
    ION 2,7:? #6;"ORIGINAL ST."
1415 POSITION 2,8:? #6;"CHARLES BRANNON"
1420 FOR I=1536 TO 1710:READ A:POKE I,A:POKE
    709,A:SOUND 0,A,10,4:NEXT I
1430 SOUND 0,0,0,0:RETURN
1440 DATA 72,169,100,141,10,212
1450 DATA 141,24,208,141,26,208
1460 DATA 169,6,141,9,212,104
1470 DATA 64,104,104,133,204,104
1480 DATA 133,203,169,0,133,205
1490 DATA 169,224,133,206,162,4
1500 DATA 160,0,177,205,145,203
1510 DATA 200,208,249,230,204,230
1520 DATA 206,202,208,240,96,104
1530 DATA 162,16,169,9,157,66
1540 DATA 3,104,157,69,3,104
1550 DATA 157,68,3,169,0,157
1560 DATA 72,3,169,4,157,73
1570 DATA 3,32,86,228,96,104

```

```

1580 DATA 162,16,169,5,76,58
1590 DATA 6,9,104,169,0,9,0,133
1600 DATA 212,169,0,133,213,96
1601 DATA 72,138,72,152,72,169,0,162,0,160,0
1602 DATA 141,10,212,141,26,208
1603 DATA 142,24,208,140,25,208
1604 DATA 169,0,141,22,208,141,10,210,169,6,
    141,9,212,169,0,141,23,208,169,156,141,
    0,2
1605 DATA 104,168,104,170,104,64,72,169,0,14
    1,10,212,141,26,208,169,104,141,10,210,
    141,0,2,104,64
1610 GOSUB 20:POSITION 14,0:?"Filename?";
1620 FN$="":K=0
1630 POKE 20,0
1640 IF PEEK(764)<255 AND PEEK(764)<>39 AND
    PEEK(764)<>60 THEN 1670
1650 IF PEEK(20)<10 THEN 1640
1660 ? CHR$(21+11*K);"{LEFT}";:K=1-K:GOTO 16
    30
1670 GET #2,A
1680 IF A=155 THEN ? " ";:FOR I=1 TO LEN(FN$
    )+10:?"{BACK S}";:NEXT I:RETURN
1690 IF A=126 AND LEN(FN$)>1 THEN FN$=FN$(1,
    LEN(FN$)-1):?"{LEFT}";CHR$(A);:GOTO 16
    30
1695 IF A=126 AND LEN(FN$)=1 THEN ? CHR$(A);
    :GOTO 1620
1700 IF A=58 OR (A>47 AND A<58) OR (A>64 AND
    A<=90) OR A=46 THEN 1720
1710 GOTO 1630
1720 IF LEN(FN$)<14 THEN FN$(LEN(FN$)+1)=CHR
    $(A):? CHR$(A);
1730 GOTO 1630
1740 END
1750 REM GET CHOICE OF CHARACTER
1760 CY=INT(MRY/32):CX=MRY-32*CY
1770 C=CX+CY*32
1780 POKE SD+CX+CY*40+4,C+128
1790 POKE ASD+CX+CY*40+4,C+128
1800 IF STRIG(0)=0 OR PEEK(764)<255 THEN MRY
    =C:GOTO 1900
1810 ST=STICK(0):IF ST=15 THEN 1800
1820 POKE 53279,0
1830 GOSUB 1900
1840 CX=CX-(ST=11)+(ST=7):CY=CY-(ST=14)+(ST=
    13)
1850 IF CX<0 THEN CX=31:CY=CY-1
1860 IF CX>31 THEN CX=0:CY=CY+1

```

```

1870 IF CY<0 THEN CY=3
1880 IF CY>3 THEN CY=0
1890 GOTO 1770
1900 POKE SD+CX+CY*40+4,C
1910 POKE ASD+CX+CY*40+4,C
1920 RETURN
2000 IF K<>ASC("A") THEN 2200
2005 POKE 54286,0
2007 POKE ASD+169+(ANTIC*10),0:POKE ASD+190+
    (ANTIC*30),0
2010 IF ANTIC=1 THEN 2100
2020 POKE DL+24,5
2030 FOR I=19 TO 23:POKE DL+I,4:NEXT I:POKE
    DL+22,132
2040 POKE 512,104:ANTIC=1
2050 COLF0=2*16+6:COLF1=6*16+6
2060 COLF2=10*16+8:COLF3=15*16+8
2070 POKE 1664,COLF0:POKE 1648,COLF1
2080 POKE 1650,COLF2:POKE 1677,COLF3
2090 POKE 54286,192:T=51:GOTO 2127
2100 ANTIC=0:POKE DL+23,6:POKE DL+24,7
2110 POKE 512,0:FOR I=19 TO 22:POKE DL+I,2:N
    EXT I
2120 POKE 54286,192:T=85
2127 GOSUB 330:POKE ASD+169+(ANTIC*10),C:POK
    E ASD+190+(ANTIC*30),C:GOTO 520
2200 IF K<>ASC("P") THEN 3000
2205 ST=0:ED=10:GOSUB 20
2210 POSITION 14,0:CT=0
2220 ? "{5 SPACES}"PRINT MODE"
2230 ? :? " Press START to return"
2240 ? "{5 SPACES}"to menu"
2250 ? :? " Press SELECT to clear"
2260 ? "{3 SPACES}"typing area"
2270 KK=PEEK(53279):IF KK=6 THEN GOSUB 390:G
    OTO 520
2280 IF KK=5 THEN 2600
2290 P=PEEK(764):IF P=255 THEN 2270
2300 GET #2,K
2302 IF K>=0 AND K<32 OR K>=128 AND K<160 TH
    EN K=K+64:GOTO 2306
2304 IF K>=32 AND K<96 OR K>=160 AND K<224 T
    HEN K=K-32
2306 IF CT>(ANTIC+1)*17 THEN 2320
2310 POKE ASD+161+CT,K:POKE ASD+181+(ANTIC*2
    0)+CT,K:CT=CT+1:GOTO 2270
2320 FOR I=0 TO 17*(ANTIC+1):POKE ASD+161+I,
    PEEK(ASD+162+I):POKE ASD+181+(ANTIC*20)
    +I,PEEK(ASD+182+(ANTIC*20)+I)

```

```

2330 NEXT I:CT=17*(ANTIC+1):GOTO 2310
2600 FOR I=0 TO 19*(ANTIC+1):POKE ASD+161+I,
  0:POKE ASD+181+(ANTIC*20)+I,0:NEXT I:CT
  =0:GOTO 2270
3000 IF K<>ASC("K") THEN 3500
3010 ST=0:ED=10:GOSUB 20:DIS=0
3020 POKE 82,14:POSITION 14,0:? "COLOR CHANG
  E MODE"
3030 ? " PRESS K TO RETURN"
3040 ? "{5 SPACES}TO MENU"
3050 ? " ☐ PLAYFIELD 0"
3060 ? " ☐ PLAYFIELD 1"
3070 ? " ☐ PLAYFIELD 2"
3080 ? " ☐ PLAYFIELD 3"
3090 ? " ☐ BACKGROUND":? "☐ READ REGISTER"
3100 GET #2,K:DIS=0:IF K=ASC("0") THEN DIS=18
3105 IF K=ASC("R") THEN RDE=1:GOTO 3100
3110 IF K=ASC("1") THEN DIS=31
3120 IF K=ASC("2") THEN DIS=2
3130 IF K=ASC("3") THEN DIS=4
3140 IF K=ASC("B") THEN DIS=48
3150 IF K=ASC("K") THEN GOSUB 390:GOTO 520
3155 IF RDE=1 THEN 3410
3160 IF DIS=0 THEN 3100
3170 ST=2:ED=10:GOSUB 20
3180 POKE 82,14:POSITION 14,0
3190 ? "☐ GREY ☐ GOLD ☐ ORANGE"
3200 ? "☐ RED{3 SPACES}☐ PINK ☐ PURPLE"
3210 ? "☐ BLUE ☐ BLUE ☐ LT.BLUE"
3220 ? "☐ TURQUOISE ☐ GREENBLUE"
3230 ? "☐ GREEN{5 SPACES}☐ YELLOW/GR"
3240 ? "☐ ORANGE/GR ☐ LT.ORANGE"
3245 TRAP 3400
3250 INPUT COL:? "{3 SPACES}Luminosity"
3260 ? " input(0-14)";
3270 INPUT LUM
3280 CLCHG=COL*16+LUM
3290 POKE 1646+DIS,CLCHG
3300 GOTO 3010
3400 TRAP 40000:POSITION 14,6:? "TRY AGAIN":
  FOR I=1 TO 100:NEXT I:POSITION 14,6:? "
  {9 SPACES}":POSITION 14,6:GOTO 3245
3410 RDE=0:DRE=PEEK(1646+DIS):POSITION 14,9:
  ? "COLOR REGISTER ";CHR$(K);"="";"
  {3 SPACES}";"{3 LEFT}";DRE:GOTO 3100
3500 IF K<>ASC("D") THEN 520
3510 POKE 53248,0:POKE 53249,0:POKE 53250,0:
  POKE 82,0:ST=0:ED=10:GOSUB 3600
3515 GOSUB 3620:N$=" "

```

```

3520 POSITION 2,0:?"First letter of FONT to
    be made":?"into DATA statements";:INP
    UT N$:A=ASC(N$):GOSUB 3700
3530 SST=A:?" Last letter of FONT";:INPUT
    N$:A=ASC(N$):GOSUB 3700:LLT=A+1:
3534 TRAP 3610
3535 ? " DATA LINE START (must be 4000 or mo
    re)":INPUT L:IF L<4000 THEN 3535
3536 DTASTART=L:ED=5:ST=0:GOSUB 3600
3540 FOR J=SST TO LLT-1:POSITION 0,2:?"
    {DOWN}";L;" D.";:FOR A=0 TO 7:?" PEEK(CH
    SET+J*8+A);";";:NEXT A:?" {LEFT}";" "
3550 ? :?"CONT":POSITION 0,0:POKE 842,13:STOP
3560 POKE 842,12:L=L+10:ED=10:ST=0:GOSUB 360
    0:NEXT J
3565 POSITION 0,0:?" SAVE FONT";:INPUT N$:I
    F N$(1,1)="Y" THEN GOSUB 1610:LIST FN$,
    DTASTART,L:GOTO 3570
3566 IF N$(1,1)<>"N" THEN GOSUB 3600:GOTO 3565
3570 ? " Finished";:INPUT N$:IF N$(1,1)="Y"
    THEN 1140
3575 IF N$(1,1)<>"N" THEN GOSUB 3600:POSITIO
    N 0,0:GOTO 3570
3580 POKE 53248,64:POKE 53249,64:POKE 53250,
    64:ED=10:ST=0:GOSUB 3600:POKE 82,2:POSI
    TION 2,0:GOSUB 380
3590 GOSUB 330:GOSUB 390:GOTO 520
3600 POSITION 0,0:FOR I=ST TO ED:?" "
    {39 SPACES}":NEXT I:RETURN
3610 TRAP 40000:GOTO 3534
3620 POSITION 0,0:?" DELETE ANY DATA LINES"
    ;:INPUT N$:IF N$(1,1)="N" THEN ED=10:ST
    =0:GOSUB 3600:RETURN
3630 IF N$(1,1)<>"Y" THEN 3620
3635 TRAP 3690
3640 ? " START LINE NUMBER";:INPUT L:?" END
    LINE NUMBER";:INPUT LLT
3645 IF L<4000 OR LLT<L THEN ED=10:ST=0:GOSU
    B 3600:POSITION 0,0:GOTO 3640
3646 ED=10:ST=0:GOSUB 3600
3650 FOR J=L TO LLT STEP 10:POSITION 0,2:?"
    {DOWN}";J:?" ? "CONT":POSITION 0,0:POKE
    842,13:STOP
3660 POKE 842,12:NEXT J:ED=10:ST=0:GOSUB 360
    0:RETURN
3690 TRAP 40000:GOSUB 3600:GOTO 3635
3700 IF A>=32 AND A<96 THEN A=A-32:RETURN
3710 IF A>=0 AND A<32 THEN A=A+64:RETURN
3720 IF A>127 THEN A=A-128:GOTO 3700
3730 RETURN

```

Program 2. Character Set Loader

```

1000 REM CHLOAD-CHARACTER SET LOADER
1005 OPEN #1,4,0,"D:FONT":REM YOUR FILENAME
    HERE
1010 X=16:CHSET=(PEEK(106)-8)*256:POKE 756,C
    HSET/256
1020 ICCOM=834:ICBADR=836:ICBLEN=840
1030 POKE ICBADR+X+1,CHSET/256:POKE ICBADR+X
    ,0
1040 POKE ICBLEN+X+1,4:POKE ICBLEN+X,0
1050 POKE ICCOM+X,7:A=USR(ADR("hhhHLV"),X):
    REM CALL CIO
1060 CLOSE #1

```

Program 3. Character Set DATAmaker

```

100 REM CHSET DATAMAKER
102 GRAPHICS 1+16:CHSET=(PEEK(106)-8)*256
105 DIM F$(14),OF$(14)
110 POSITION 3,0: ? #6;"character set"
120 POSITION 5,2: ? #6;"datamaker"
130 ? #6: ? #6;"THIS UTILITY CREATES"
140 ? #6;"A SET OF DATA STATE-"
150 ? #6;"MENTS FROM A SAVED"
160 ? #6;"CHARACTER SET. IT"
170 ? #6;"OPTIMIZES BY ONLY"
180 ? #6;"LISTING CHARACTERS"
190 ? #6;"NOT PRESENT IN THE"
200 ? #6;"STANDARD CHARACTER"
210 ? #6;"SET."
220 ? #6: ? #6;"PRESS OPTION"
230 IF PEEK(53279)<>3 THEN 230
240 GRAPHICS 1+16
250 ? #6;"THE DATA STATEMENTS"
260 ? #6;"WILL BE WRITTEN"
270 ? #6;"AS A list FILE."
280 ? #6;"USE enter TO MERGE"
290 ? #6;"THE DATA WITH YOUR"
300 ? #6;"PROGRAM.": ? #6: ? #6;"ENTER FILENAM"
    E": ? #6;"OF CHARACTER SET"
305 POKE 82,0:POKE 87,0
310 ? "{UP}{DEL LINE}";:INPUT F$:IF F$="" TH
    EN 310
315 IF F$="C" OR F$="C:" THEN CASS=1:GOTO 33
    5
320 ? "{6 UP}{6 DEL LINE}ENTER OUTPUT"
    {8 SPACES}FILENAME": ? : ?
330 ? "{UP}{DEL LINE}";:INPUT OF$:IF OF$=""
    THEN 330

```

```

335 ? "{3 UP}{3 DEL LINE}ENTER LINE # FOR
      {5 SPACES}DATA STATEMENTS":? :?
340 INPUT SLINE
345 CLOSE #1
350 GRAPHICS 2+16:SETCOLOR 4,3,0
360 IF CASS THEN ? #6:? #6;"POSITION CHARACT
      ER":? #6;"SET TAPE,HIT RETURN"
370 POSITION 5,6:? #6;"working{3 N}"
375 GOSUB 1000:REM LOAD CHARACTER SET
377 IF CASS THEN ? #6;"{CLEAR}INSERT OUTPUT
      TAPE,":? #6;"PRESS RETURN"
380 OPEN #2,8,0,OF$:POSITION 5,6:? #6;"worki
      ng{3 N}"
381 ? #2;SLINE;"CHSET=(PEEK(106)-8)*256:FOR
      I=0 TO 1023:POKE CHSET+I,PEEK(57344+I):N
      EXT I"
382 ? #2;SLINE+1;"RESTORE ";SLINE+5
383 ? #2;SLINE+2;"READ A:IF A=-1 THEN RETURN
      "
384 ? #2;SLINE+3;"FOR J=0 TO 7:READ B:POKE C
      HSET+A*8+J,B:NEXT J"
385 ? #2;SLINE+4;"GOTO ";SLINE+2
387 LINE=SLINE+4
390 FOR I=0 TO 127:F=0
400 FOR J=0 TO 7
420 IF PEEK(CHSET+I*8+J)<>PEEK(57344+I*8+J)
      THEN F=1
430 NEXT J
440 IF NOT F THEN 460
445 LINE=LINE+1
450 ? #2;LINE;" DATA ";;? #2;I;;FOR J=0 TO 7
      :? #2;",";PEEK(CHSET+I*8+J);:NEXT J:? #2
460 NEXT I:? #2;LINE+1;"DATA -1"
470 POKE 82,2:? "All finished! Use ENTER ";
      CHR$(34);OF$
480 ? "to merge the file."
490 NEW
1000 REM HIGH-SPEED LOAD OF CHARACTER SET
1005 OPEN #1,4,0,F$:REM OPEN FILE
1010 X=16:REM #10
1020 ICCOM=834:ICBADR=836:ICBLEN=840
1030 POKE ICBADR+X+1,CHSET/256:POKE ICBADR+X
      ,0
1040 POKE ICBLEN+X+1,4:POKE ICBLEN+X,0
1050 POKE ICCOM+X,7:A=USR(ADR("hhh3LV"),X):
      REM CALL CIO
1060 CLOSE #1:RETURN

```


Super TextPlot

Donald L. Vossler

This modified version of Charles Brannon's "TextPlot" runs more slowly, but adds many features for fancy text displays.

"Super TextPlot" is a machine language utility that lets you plot character images in any Atari graphics or text mode. The idea for the program was inspired by Charles Brannon's "TextPlot" utility (*COMPUTE's First Book of Atari Graphics*). Super TextPlot provides the following capabilities.

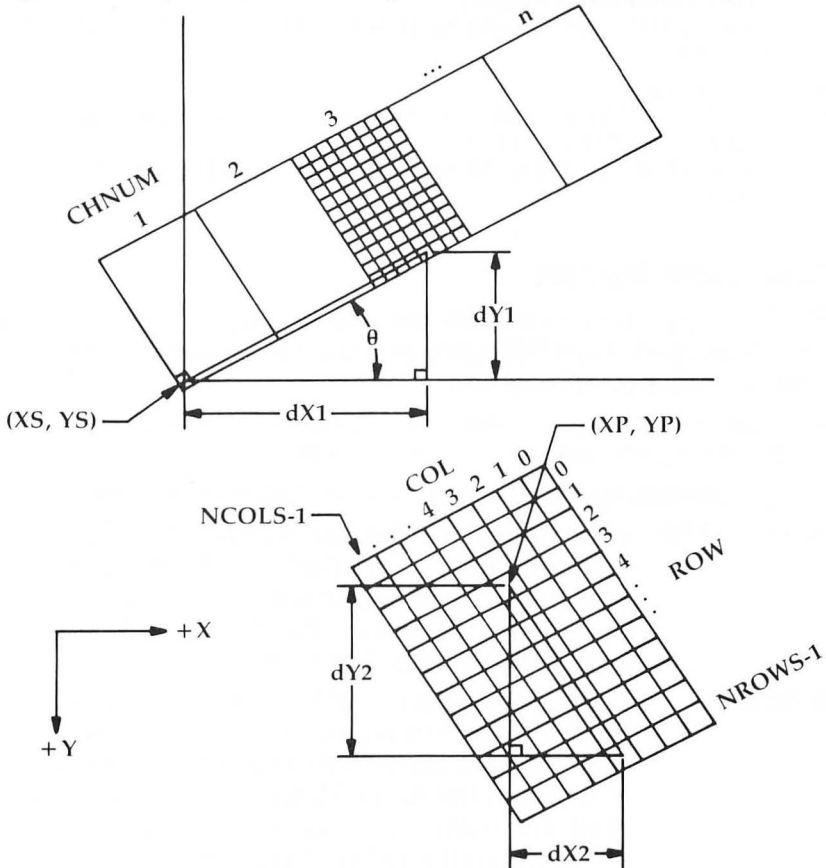
1. Plots the entire ATASCII character set, including upper/lower-case, graphics characters, special symbols, and the reverse video version of each of these characters in any graphics or text mode. Alternate character sets may be plotted by changing the CHBAS vector (location 756) to point to the alternate character set.
2. Allows the user to specify a string of characters to plot. The only length limitation for the string is that it must fit in the display area when it is plotted.
3. Allows the user to specify the starting position of the string to plot. This position can be any (X,Y) coordinate on the display.
4. Gives the user the option of overwriting the graphics already on the screen or of merging the plotted characters with the existing graphics.
5. Allows the user to select which color registers are to be used for the foreground and background of the characters plotted.
6. Allows the user to scale each character string independently in the horizontal and vertical directions by specifying the number of rows and columns for each character. The actual size of each character varies with the pixel size of the graphics mode selected. Many different-sized characters can be plotted on the same graphics screen.
7. Allows the user to select one of four angular orientations to plot each character string. The four available orientations are 90 degree increments from the horizontal.

All of these capabilities are available using one simple invocation of a machine language routine from the USR function in BASIC.

Underlying Mathematical Concepts

The fundamental trigonometric relationships used by Super TextPlot are illustrated in Figure 1. The angle THETA (θ) is measured from the horizontal +X axis to the baseline of the character string to be plotted; CHNUM is the index number of each character in the string; NROWS and NCOLS are the total number of rows and columns, respectively, to be plotted for each character; ROW and COL are the particular row and column of the pixel to

Figure 1. Super TextPlot Trigonometric Relationships



be plotted; XS and YS are the coordinates of the lower-left corner of the first character to be plotted (before the string is rotated). Using these definitions, the appropriate formulas for the point to be plotted (XP, YP) are the following:

$$XP = XS + \cos(\theta) * (CHNUM * NCOLS - 1 - COL) - \sin(\theta) * (NROWS - 1 - ROW)$$

$$YP = YS - \sin(\theta) * (CHNUM * NCOLS - 1 - COL) - \cos(\theta) * (NROWS - 1 - ROW)$$

The derivation of these formulas is shown in Figure 2.

Figure 2. Derivation of Plotting Formulas

$$XP = XS + dX1 - dX2$$

$$XP = XS + \cos(THETA) * ((CHNUM - 1) * NCOLS + (NCOLS - 1 - COL)) - \sin(THETA) * (NROWS - 1 - ROW)$$

$$XP = XS + \cos(THETA) * CHNUM * NCOLS - 1 - COL - \sin(THETA) * (NROWS - 1 - ROW)$$

$$YP = YS - dY1 - dY2$$

$$YP = YS - \sin(THETA) * ((CHNUM - 1) * NCOLS + (NCOLS - 1 - COL)) - \cos(THETA) * (NROWS - 1 - ROW)$$

$$YP = YS - \sin(THETA) * CHNUM * NCOLS - 1 - COL - \cos(THETA) * (NROWS - 1 - ROW)$$

Using Super TextPlot

With the appropriate formulas derived, the Super TextPlot routine was developed. The USR function is used to invoke the utility. The syntax for this function is:

```
A=USR (ADR (ASM$) , ADR (S$) , LEN (S$) , XS , Y
S , ORIENT , NROWS , NCOLS , FCR , BCR , PRIOR)
```

The parameters specified above have the following meanings:

ADR(ASM\$) This parameter is the starting address of the Super TextPlot routine. Since the loader for the routine uses a character string (ASM\$) to reserve space in memory for the routine, the starting address is merely the address of this string.

ADR(S\$) This parameter is the address of the string to be plotted. Usually it will be the value returned by the ADR function for the string since this is the first character in the string. However, any address is valid. For example, the address could point to a sub-string contained in a long string.

- LEN(S\$)** This parameter specifies how many characters are to be plotted. The LEN function provides the appropriate value if the entire string is to be plotted. Other values may be appropriate for plotting sub-strings. If this parameter is zero, nothing is plotted, and the USR function simply returns to the BASIC program.
- XS,YS** These two parameters specify the (X,Y) coordinates of the starting position of the string to be plotted (lower-left corner of the first character). This point is also used as the pivot point when the string is rotated (see ORIENT parameter). (XS,YS) must define a point within the limits of the current graphics mode.
- ORIENT** This parameter specifies the angular orientation of the character string to be plotted. The string is rotated counterclockwise from the horizontal +X axis about the point (XS,YS). The parameter ORIENT should be specified as an integer which is interpreted as follows:
- ORIENT = 0, 0 degree rotation
 - = 1, 90 degree rotation
 - = 2, 180 degree rotation
 - = 3, 270 degree rotation
- The value of ORIENT is interpreted MOD(3) so that ORIENT = 4 is the same as ORIENT = 0, ORIENT = 5 is the same as ORIENT = 1, etc. The high byte of the two-byte integer passed by the USR function to the machine language routine is ignored. Figure 3 illustrates the orientation of strings plotted at each of the four orientations.
- NROWS** The parameter specifies how many rows per character are to be plotted and therefore determines the height of each character. Normally, NROWS is greater than or equal to eight; however, positive values less than eight are valid and will result in characters plotted with "missing" rows. This may be useful for crowding strings into a limited space, or it may simply produce unreadable characters. If NROWS is zero, nothing is plotted, and the USR

function returns to the BASIC program. The maximum acceptable value for NROWS is 255 (the high byte of the two-byte integer passed to the machine language routine by the USR function is ignored).

- NCOLS** This parameter specifies how many columns per character are to be plotted and therefore determines the width of each character. The restrictions on the range of values for this parameter are the same as those specified for the NROWS parameter.
- FCR** This parameter specifies the foreground color register to be used when plotting the string. This indirectly specifies the color of the characters plotted in the framework of the standard SETCOLOR-COLOR concept embodied in the Atari BASIC language. In text modes (GRAPHICS 0-2) this parameter should be specified as an ATASCII code. Using Super TextPlot in this manner allows block printing of character images which are typically used as headers to identify printed listings. For example, FCR = 160 would use the reverse video space to plot large characters in GRAPHICS 0.
- BCR** This parameter specifies the background color register for each character. The comments regarding the use of FCR in text modes also apply for this parameter. If the value of the parameter PRIOR (see below) is zero, then the BCR parameter has no effect on the characters plotted.
- PRIOR** This parameter specifies the priority of the background of the character string plotted. If PRIOR is zero, the background of the characters is not plotted and existing graphics on the screen will not be disturbed. If PRIOR is a positive value, the color specified indirectly by BCR is plotted for the background (this color may be black).

The following items should be noted in relation to specifying these parameters:

1. The Super TextPlot routine does not check to make sure that

points plotted to form a string fall within the bounds of the display area. The user must insure that all the points to be plotted will fall within the display limits. Plotting points which are out of range usually results in a system crash.

2. Reverse video characters may be plotted by two different methods:

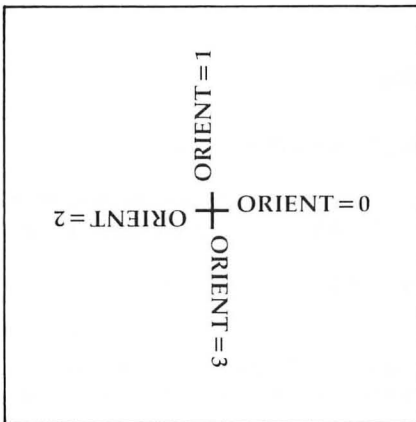
- a. Specify reverse video characters in the string to be plotted; or
- b. Specify normal characters in the string and reverse the values for FCR and BCR.

3. If the parameters FCR and BCR are assigned the same value (and PRIOR is positive), the string will be plotted but will appear as contiguously colored blocks.

4. If an improper number of parameters is specified in the USR function statement, Super TextPlot will return to the BASIC program but take no other action.

5. The value A returned by the USR function has no significance.

Figure 3. Angular Orientations of Character Strings



Loading Super TextPlot

One of the problems associated with writing utility routines in machine language is determining a safe range of memory locations which can be used to store the routine. This problem is complicated by various available memory configurations,

memory used by custom display lists, player/missile graphics, and other machine language routines.

Super TextPlot solves all of these problems by providing the machine code in a relocatable format. All of the addresses in the DATA statements are relative addresses offset from the beginning of the routine. These addresses are flagged as minus numbers in the DATA statements. When the loader routine is invoked, it reserves a character string (ASM\$) in which the machine code is stored. As each instruction code is loaded into this string, the addresses are modified to reflect the actual memory locations utilized.

Applications for Super TextPlot

Since Super TextPlot is a utility program, it can be treated as an extension to the BASIC programming language and therefore becomes one of the tools available to a programmer. Obvious examples for the use of this routine include labelling graphs and bar charts, adding text to graphic game displays, and developing colorful and attractive message displays. Super TextPlot can be an effective marketing/sales tool. A variety of textual messages can be displayed on a demonstration computer system in order to attract customers and provide information in an eye-catching format.

Super TextPlot Demonstration

```

1000 REM
1010 REM -----INITIALIZATION-----
1020 REM
1030 DIM S$(40):DEG:?"LOADING ASSEMBLY COD
    E":?"45 SECOND DELAY ...":GOSUB 8000
1040 REM
1050 REM -----DEMO #1-----
1060 REM
1070 GRAPHICS 7+16
1080 S$="SUPER TEXTPLOT":XS=24:YS=24:ORIENT=
    0:NROWS=24:NCOLS=8:FCR=3:BCR=0:PRIOR=0:
    GOSUB 8000
1090 S$="FOR":XS=68:YS=44:ORIENT=0:NROWS=8:N
    COLS=8:FCR=1:BCR=0:PRIOR=0:GOSUB 8000
1100 S$="ATARI":XS=0:YS=95:ORIENT=0:NROWS=32
    :NCOLS=32:FCR=2:BCR=0:PRIOR=0:GOSUB 800
    0
1110 S$="COMPUTE!":NROWS=8:NCOLS=8:FCR=3:BCR
    =1:PRIOR=1
1120 XS=7:YS=64:ORIENT=1:PRIOR=1:GOSUB 8000

```

```

1130 XS=151:YS=2:ORIENT=3:GOSUB 8000
1140 GOSUB 7000
1150 REM
1800 REM -----DEMO #2-----
1810 REM
1820 GRAPHICS 7+16
1830 COLOR 3
1840 PLOT 34,2:DRAWTO 126,2:DRAWTO 126,94:DR
    AWTO 34,94:DRAWTO 34,2
1850 PLOT 51,19:DRAWTO 109,19:DRAWTO 109,77:
    DRAWTO 51,77:DRAWTO 51,19
1860 PLOT 60,28:DRAWTO 100,28:DRAWTO 100,68:
    DRAWTO 60,68:DRAWTO 60,28
1870 PLOT 34,2:DRAWTO 60,28
1880 PLOT 126,2:DRAWTO 100,28
1890 PLOT 126,94:DRAWTO 100,68
1900 PLOT 34,94:DRAWTO 60,68
1910 S$="ATARI":FCR=2:BCR=0:NROWS=8:NCOLS=8:
    PRIOR=0
1920 XS=60:YS=27:ORIENT=0:GOSUB 8000
1930 XS=101:YS=28:ORIENT=3:GOSUB 8000
1940 XS=98:YS=69:ORIENT=2:GOSUB 8000
1950 XS=58:YS=67:ORIENT=1:GOSUB 8000
1960 NROWS=16:NCOLS=11:FCR=1
1970 XS=53:YS=18:ORIENT=0:GOSUB 8000
1980 XS=110:YS=21:ORIENT=3:GOSUB 8000
1990 XS=107:YS=78:ORIENT=2:GOSUB 8000
2000 XS=50:YS=75:ORIENT=1:GOSUB 8000
2010 XS=61:YS=67:ORIENT=0:FCR=3:BCR=2:NROWS=
    39:NCOLS=39:PRIOR=1
2020 FOR I=1 TO 8
2030 S$="COMPUTE!":S$=S$(I,I):GOSUB 8000
2040 NEXT I
2050 GOSUB 7000
2060 GOTO 2060
7000 REM
7010 REM ----COLOR FLASH ROUTINE----
7020 REM
7030 FOR I=1 TO 50:FOR J=0 TO 2:SETCOLOR J,R
    ND(0)*16,RND(0)*16:FOR W=1 TO 5:NEXT W:
    NEXT J:NEXT I:RETURN
7997 REM
7998 REM -- SUPER TEXTPLOT ROUTINE --
7999 REM
8000 IF ASMLD=1 THEN A=USR(ADR(ASM$),ADR(S$)
    ,LEN(S$),XS,YS,ORIENT,NROWS,NCOLS,FCR,B
    CR,PRIOR):RETURN
8010 ASMLD=1
8020 DIM ASM$(725)

```



```

8030 FOR I=ADR(ASM$) TO ADR(ASM$)+724
8040 READ A
8050 ON (SGN(A)+2) GOSUB 8080,8160,8220
8060 NEXT I
8070 GOTO 8000
8080 READ B
8090 ADDR=ABS(A)+256*ABS(B)+ADR(ASM$)
8100 ADDRHI=INT(ADDR/256)
8110 ADDRLO=ADDR-256*ADDRHI
8120 POKE I,ADDRLO
8130 POKE I+1,ADDRHI
8140 I=I+1
8150 RETURN
8160 READ B
8170 IF B<0 THEN 8090
8180 POKE I,A
8190 POKE I+1,B
8200 I=I+1
8210 RETURN
8220 POKE I,A
8230 RETURN
8240 DATA 104,141,-255,0,10,170,240,8
8250 DATA 104,157,-255,0,202,76,-6,0
8260 DATA 173,-255,0,201,10,240,1,96
8270 DATA 173,-16,-1,208,1,96,173,-8
8280 DATA -1,208,1,96,173,-6,-1,208,1
8290 DATA 96,173,-10,-1,41,3,141,-10
8300 DATA -1,173,-18,-1,133,203,173
8310 DATA -19,-1,133,204,169,0,141
8320 DATA -17,-1,238,-17,-1,56,173
8330 DATA -16,-1,237,-17,-1,16,3,76
8340 DATA -254,0,32,-32,-1,173,-6,-1
8350 DATA 141,-7,-1,206,-7,-1,174,-7
8360 DATA -1,224,255,208,3,76,-251,0
8370 DATA 172,-6,-1,32,-138,-1,140,-3
8380 DATA -1,173,-6,-1,174,-17,-1,172
8390 DATA -7,-1,32,-195,-1,140,-20,-1
8400 DATA 142,-21,-1,169,255,141,-9
8410 DATA -1,238,-9,-1,174,-9,-1,236
8420 DATA -8,-1,208,3,76,-248,0,172
8430 DATA -8,-1,32,-138,-1,140,-5,-1
8440 DATA 172,-5,-1,177,205,172,-3,-1
8450 DATA 57,-24,-1,240,2,169,1,141
8460 DATA -1,-1,173,-135,-1,240,9,56
8470 DATA 169,1,237,-1,-1,141,-1,-1
8480 DATA 173,-1,-1,208,5,173,0,-1
8490 DATA 240,46,174,-2,-1,173,-1,-1
8500 DATA 240,3,174,-4,-1,142,-212,-2
8510 DATA 173,-8,-1,162,1,172,-9,-1

```

```

8520 DATA 32,-195,-1,140,-22,-1,142
8530 DATA -23,-1,32,-16,-2,173,-184
8540 DATA -2,174,-183,-2,172,-185,-2
8550 DATA 32,-187,-2,76,-134,0,76,-89
8560 DATA 0,76,-65,0,96,0,0,0,0,0,0
8570 DATA 0,0,0,0,0,0,0,0,0,0,0,0
8580 DATA 0,0,0,0,1,2,4,8,16,32,64
8590 DATA 128,169,0,141,-135,-1,172
8600 DATA -17,-1,136,177,203,141,-136
8610 DATA -1,16,13,169,1,141,-135,-1
8620 DATA 173,-136,-1,41,127,141,-136
8630 DATA -1,56,173,-136,-1,233,32,16
8640 DATA 12,24,173,-136,-1,105,64
8650 DATA 141,-137,-1,76,-107,-1,56
8660 DATA 173,-136,-1,233,96,16,12,56
8670 DATA 173,-136,-1,233,32,141,-137
8680 DATA -1,76,-107,-1,173,-136,-1
8690 DATA 141,-137,-1,169,0,133,206
8700 DATA 173,-137,-1,133,205,162,3
8710 DATA 24,38,205,38,206,202,208
8720 DATA 248,24,165,206,109,244,2
8730 DATA 133,206,96,0,0,0,142,-193
8740 DATA -1,140,-194,-1,142,-191,-1
8750 DATA 169,0,141,-192,-1,162,3,24
8760 DATA 46,-191,-1,46,-192,-1,202
8770 DATA 208,246,160,255,200,56,173
8780 DATA -191,-1,237,-194,-1,141
8790 DATA -191,-1,173,-192,-1,233,0
8800 DATA 141,-192,-1,16,235,142,-191
8810 DATA -1,96,0,0,0,0,141,-12,-2
8820 DATA 140,-13,-2,169,0,141,-14,-2
8830 DATA 141,-15,-2,24,173,-12,-2
8840 DATA 109,-14,-2,141,-14,-2,169,0
8850 DATA 109,-15,-2,141,-15,-2,202
8860 DATA 208,235,56,173,-14,-2,233,1
8870 DATA 141,-14,-2,173,-15,-2,233,0
8880 DATA 141,-15,-2,56,173,-14,-2
8890 DATA 237,-13,-2,141,-14,-2,168
8900 DATA 173,-15,-2,233,0,141,-15,-2
8910 DATA 170,96,0,0,0,0,174,-10,-1
8920 DATA 208,39,24,173,-14,-1,109
8930 DATA -20,-1,141,-183,-2,173,-15
8940 DATA -1,109,-21,-1,141,-184,-2
8950 DATA 56,173,-12,-1,237,-22,-1
8960 DATA 141,-185,-2,173,-13,-1,237
8970 DATA -23,-1,141,-186,-2,96,202
8980 DATA 208,39,56,173,-14,-1,237
8990 DATA -22,-1,141,-183,-2,173,-15
9000 DATA -1,237,-23,-1,141,-184,-2

```

```
9010 DATA 56,173,-12,-1,237,-20,-1
9020 DATA 141,-185,-2,173,-13,-1,237
9030 DATA -21,-1,141,-186,-2,96,202
9040 DATA 208,39,56,173,-14,-1,237
9050 DATA -20,-1,141,-183,-2,173,-15
9060 DATA -1,237,-21,-1,141,-184,-2
9070 DATA 24,173,-12,-1,109,-22,-1
9080 DATA 141,-185,-2,173,-13,-1,109
9090 DATA -23,-1,141,-186,-2,96,24
9100 DATA 173,-14,-1,109,-22,-1,141
9110 DATA -183,-2,173,-15,-1,109,-23
9120 DATA -1,141,-186,-2,24,173,-12
9130 DATA -1,109,-20,-1,141,-185,-2
9140 DATA 173,-13,-1,109,-21,-1,141
9150 DATA -186,-2,96,0,0,0,0,134,85
9160 DATA 133,86,132,84,162,96,169,11
9170 DATA 157,66,3,169,0,157,72,3,173
9180 DATA -212,-2,32,86,228,96,1
```

Circles

Jeffrey S. McArthur

Every Atari graphics programmer needs to draw circles. This tutorial will show you how to draw a circle—and draw one fast—without jumping through hoops. There are several drawing utilities here, from an elementary BASIC routine which takes 60 seconds to a machine language version that finishes in a fraction of a second.

Program 1 draws circles, but takes more than a minute to draw a circle, no matter how big or small it is.

Reflections

A circle is symmetrical, so why don't we take advantage of its symmetry? If we know the value of one point, we can reflect it across the X-axis or across the Y-axis. That is, if we know (X,Y) is a point on the circle, then so is (X,-Y). The same is true for (-X,Y) and (-X,-Y). So we have to do only a quarter of the work. Circles are also symmetrical along the X = Y line. If we know (X,Y) is on the circle, then so is (Y,X). Now we have to find only an eighth of the points. Program 2 uses that method.

Unfortunately, even doing only one-eighth of the work, we still need more than 10 seconds to draw the circle. Perhaps there is a better way. Instead of using sines and cosines, use the equation:

$$X^2 + Y^2 = R^2$$

That isn't very useful, but we can rearrange the equation and get:

$$Y = \sqrt{R^2 - X^2}$$

So all we have to do is find Y for X = -R to R. However, since the square root function returns only the positive square root, we also have to plot the negative square root. Program 3 is an example of how to do that. This method is faster than using sines or cosines, but it still takes more than 16 seconds. So using Program 4, we reflect it, like we did in Program 2.

Now we have a method that takes only five seconds on a large circle and is a lot faster on the smaller ones. If you take a close look at how Program 4 draws the circle, you see it draws lines of different lengths. This method works fine on a screen, but on a plotter the circle has flat spots.

A Faster Circle

The screen is made up of an array of points. Each point is addressed by two coordinates (X,Y). However, X and Y are *always* integers. In Atari BASIC you can PLOT 0.5,0.5, but the points are rounded to integers. So if you are at one point on the circle and are trying to figure where the next point is, you can go in eight directions.

If you divide the circle into quarters, then only three of those directions are valid. If you divide the circle into eight parts, you can go in only two directions. For example, if you are on the circle at (R,0), the next point is either (R-1,0) or (R-1,1). This method is called a *potential function*. Since the screen cannot plot points except with integers, there is a small error that is not always equal to zero.

We want to keep the error as small as possible. We also reflect it eight ways as before. That takes only three seconds, and we never have to draw any long lines. Program 5 uses this method.

Notice also that you can achieve the entire result using only addition and subtraction. Such programs can be easily converted to machine language since we don't have to multiply or divide. Program 7 is a machine language program to draw a circle. Program 6 calls the machine language and takes less than 2/10 second to draw a circle.

The machine language is called by a USR function. The parameters that are passed to it are, in order: the address of the code, the X coordinate of the center of the circle, the Y coordinate of the center of the circle, the radius, and the mode of drawing. The mode of drawing means

- 0: turn point off
- 1: turn point on
- 2: invert point

The only problem with the machine language program is that it does no checking to see if the circle goes off the screen. And no clipping is done. Therefore, if your circle goes off the screen, you will write over other memory.

Program 1. Sines and Cosines

```
100 REM CIRCLE DEMONSTRATION
110 REM PROGRAM #1
120 REM
130 REM
140 REM THIS METHOD TAKES APPROXIMATELY 61 S
    ECNDS
```

```
200 DEG
210 GRAPHICS 8
220 COLOR 1
230 SETCOLOR 2,0,0
240 A=160
250 B=80
260 R=50
300 FOR ALPHA=0 TO 360
310 X1=INT(R*COS(ALPHA)+0.5)
320 Y1=INT(R*SIN(ALPHA)+0.5)
330 PLOT A+X1,B+Y1
340 NEXT ALPHA
```

Program 2. Sines and Cosines Reflected

```
100 REM CIRCLE DEMONSTRATION
110 REM PROGRAM #2
120 REM
130 REM
140 REM THIS METHOD TAKES APPROXIMATELY 11 S
    ECONDS
200 DEG
210 GRAPHICS 8
220 COLOR 1
230 SETCOLOR 2,0,0
240 A=160
250 B=80
260 R=50
270 PLOT A+R,B
300 FOR ALPHA=0 TO 45
310 X1=INT(R*COS(ALPHA)+0.5)
320 Y1=INT(R*SIN(ALPHA)+0.5)
330 PLOT A+X1,B+Y1
340 PLOT A-X1,B+Y1
350 PLOT A+X1,B-Y1
360 PLOT A-X1,B-Y1
370 PLOT A+Y1,B+X1
380 PLOT A-Y1,B+X1
390 PLOT A+Y1,B-X1
400 PLOT A-Y1,B-X1
410 NEXT ALPHA
```

Program 3. Square Root

```
100 REM CIRCLE DEMONSTRATION
110 REM PROGRAM #3
120 REM
130 REM
140 REM THIS METHOD TAKES APPROXIMATELY 17 S
    ECONDS
210 GRAPHICS 8
```

```

220 COLOR 1
230 SETCOLOR 2,0,0
240 A=160
250 B=80
260 R=50
270 X0=-R:Y0=0
300 FOR X1=-R TO R
310 Y1=INT(0.5+SQR(R*R-X1*X1))
330 PLOT A+X0,B+Y0:DRAWTO A+X1,B+Y1
335 PLOT A+X0,B-Y0:DRAWTO A+X1,B-Y1
336 X0=X1:Y0=Y1
340 NEXT X1

```

Program 4. Square Root Reflected

```

100 REM CIRCLE DEMONSTRATION
110 REM PROGRAM #4
120 REM
130 REM
140 REM THIS METHOD TAKES APPROXIMATELY 5 SE
    CONDS
210 GRAPHICS 8
220 COLOR 1
230 SETCOLOR 2,0,0
240 A=160
250 B=80
260 R=50
270 X0=-R:Y0=0
280 X1=-R
290 Y1=INT(0.5+SQR(R*R-X1*X1))
300 PLOT A+X0,B+Y0:DRAWTO A+X1,B+Y1
310 PLOT A-X0,B+Y0:DRAWTO A-X1,B+Y1
320 PLOT A+X0,B-Y0:DRAWTO A+X1,B-Y1
330 PLOT A-X0,B-Y0:DRAWTO A-X1,B-Y1
340 PLOT A+Y0,B+X0:DRAWTO A+Y1,B+X1
350 PLOT A-Y0,B+X0:DRAWTO A-Y1,B+X1
360 PLOT A+Y0,B-X0:DRAWTO A+Y1,B-X1
370 PLOT A-Y0,B-X0:DRAWTO A-Y1,B-X1
380 X0=X1:Y0=Y1
390 IF -X1>=Y1 THEN X1=X1+1:GOTO 290

```

Program 5. Potential

```

100 REM CIRCLE DEMONSTRATION
110 REM PROGRAM #5
120 REM
130 REM
140 REM THIS METHOD TAKES APPROXIMATELY 3 SE
    CONDS
210 GRAPHICS 8

```

```

220 COLOR 1
230 SETCOLOR 2,0,0
240 A=160
250 B=80
260 R=50
270 PHI=0
280 Y1=0
290 X1=R
300 PHIY=PHI+Y1+Y1+1
310 PHIXY=PHIY-X1-X1+1
400 PLOT A+X1,B+Y1
410 PLOT A-X1,B+Y1
420 PLOT A+X1,B-Y1
430 PLOT A-X1,B-Y1
440 PLOT A+Y1,B+X1
450 PLOT A-Y1,B+X1
460 PLOT A+Y1,B-X1
470 PLOT A-Y1,B-X1
500 PHI=PHIY
510 Y1=Y1+1
520 IF ABS(PHIXY)<ABS(PHIY) THEN PHI=PHIXY:X
    1=X1-1
530 IF X1>=Y1 THEN 300

```

Program 6. BASIC Call to Machine Language

```

100 REM CIRCLE DEMONSTRATION
110 REM PROGRAM #6
120 REM
130 REM
140 REM THIS METHOD TAKES APPROXIMATELY 0.18
    33 SECONDS
210 GRAPHICS 8
220 COLOR 1
230 SETCOLOR 2,0,0
240 A=160
250 B=80
260 R=50
270 P=7*16*16*16
300 I=USR(P,A,B,R,1)

```

Program 7. Machine Language Circle Drawing Subroutine

```

10 REM 28000- IS SUBROUTINE
20 GOSUB 28000
30 END
28000 FOR I=0 TO 758:READ A:POKE 28672+I,A:N
    EXT I
28004 RESTORE 29500
28005 FOR I=1577 TO 1584:READ A:POKE I,A:NEX
    T I

```



```

28010 RETURN
28672 DATA 104,104,141,5,6,104
28678 DATA 141,4,6,104,141,7
28684 DATA 6,104,141,6,6,104
28690 DATA 141,9,6,141,12,6
28696 DATA 104,141,8,6,141,11
28702 DATA 6,104,104,141,10,6
28708 DATA 201,3,144,1,96,169
28714 DATA 0,141,13,6,141,14
28720 DATA 6,141,15,6,141,16
28726 DATA 6,24,173,4,6,109
28732 DATA 11,6,141,25,6,173
28738 DATA 5,6,109,12,6,141
28744 DATA 26,6,24,173,4,6
28750 DATA 109,13,6,141,29,6
28756 DATA 173,5,6,109,14,6
28762 DATA 141,30,6,56,173,4
28768 DATA 6,237,11,6,141,27
28774 DATA 6,173,5,6,237,12
28780 DATA 6,141,28,6,56,173
28786 DATA 4,6,237,13,6,141
28792 DATA 31,6,173,5,6,141
28798 DATA 14,6,141,32,6,24
28804 DATA 173,6,6,109,11,6
28810 DATA 141,33,6,173,7,6
28816 DATA 109,12,6,141,34,6
28822 DATA 24,173,6,6,109,13
28828 DATA 6,141,37,6,173,7
28834 DATA 6,109,14,6,141,38
28840 DATA 6,56,173,6,6,237
28846 DATA 11,6,141,35,6,173
28852 DATA 7,6,237,12,6,141
28858 DATA 36,6,56,173,6,6
28864 DATA 237,13,6,141,39,6
28870 DATA 173,7,6,237,14,6
28876 DATA 141,40,6,173,25,6
28882 DATA 141,0,6,173,26,6
28888 DATA 141,1,6,173,37,6
28894 DATA 141,2,6,173,38,6
28900 DATA 141,3,6,32,106,114
28906 DATA 173,27,6,141,0,6
28912 DATA 173,28,6,141,1,6
28918 DATA 32,106,114,173,25,6
28924 DATA 141,0,6,173,26,6
28930 DATA 141,1,6,173,39,6
28936 DATA 141,2,6,173,40,6
28942 DATA 141,3,6,32,106,114
28948 DATA 173,27,6,141,0,6
28954 DATA 173,28,6,141,1,6

```

28960 DATA 32,106,114,173,29,6
28966 DATA 141,0,6,173,30,6
28972 DATA 141,1,6,173,33,6
28978 DATA 141,2,6,173,34,6
28984 DATA 141,3,6,32,106,114
28990 DATA 173,31,6,141,0,6
28996 DATA 173,32,6,141,1,6
29002 DATA 32,106,114,173,29,6
29008 DATA 141,0,6,173,30,6
29014 DATA 141,1,6,173,35,6
29020 DATA 141,2,6,173,36,6
29026 DATA 141,3,6,32,106,114
29032 DATA 173,31,6,141,0,6
29038 DATA 173,32,6,141,1,6
29044 DATA 32,106,114,173,14,6
29050 DATA 205,12,6,240,3,144
29056 DATA 10,96,173,13,6,205
29062 DATA 11,6,144,1,96,173
29068 DATA 11,6,133,4,173,12
29074 DATA 6,133,5,173,13,6
29080 DATA 133,205,173,14,6,133
29086 DATA 206,6,4,38,5,6
29092 DATA 205,38,206,56,165,205
29098 DATA 109,15,6,141,17,6
29104 DATA 165,206,109,16,6,141
29110 DATA 18,6,24,173,17,6
29116 DATA 229,4,141,19,6,173
29122 DATA 18,6,229,5,141,20
29128 DATA 6,173,18,6,16,27
29134 DATA 73,255,141,22,6,173
29140 DATA 17,6,73,255,24,105
29146 DATA 1,141,21,6,173,22
29152 DATA 6,105,0,141,22,6
29158 DATA 24,144,9,141,22,6
29164 DATA 173,17,6,141,21,6
29170 DATA 173,20,6,16,27,73
29176 DATA 255,141,24,6,173,19
29182 DATA 6,73,255,24,105,1
29188 DATA 141,23,6,173,24,6
29194 DATA 105,0,141,24,6,24
29200 DATA 144,9,141,24,6,173
29206 DATA 19,6,141,23,6,173
29212 DATA 17,6,141,15,6,173
29218 DATA 18,6,141,16,6,24
29224 DATA 173,13,6,105,1,141
29230 DATA 13,6,173,14,6,105
29236 DATA 0,141,14,6,173,22
29242 DATA 6,205,24,6,144,39
29248 DATA 208,8,173,21,6,205

29254 DATA 23,6,144,29,173,19
29260 DATA 6,141,15,6,173,20
29266 DATA 6,141,16,6,56,173
29272 DATA 11,6,233,1,141,11
29278 DATA 6,173,12,6,233,0
29284 DATA 141,12,6,76,55,112
29290 DATA 173,2,6,133,205,169
29296 DATA 0,133,206,6,205,38
29302 DATA 206,6,205,38,206,6
29308 DATA 205,38,206,165,205,133
29314 DATA 4,165,206,133,5,6
29320 DATA 205,38,206,6,205,38
29326 DATA 206,24,165,205,101,4
29332 DATA 133,205,165,206,101,5
29338 DATA 133,206,173,0,6,133
29344 DATA 4,173,1,6,133,5
29350 DATA 70,5,102,4,70,5
29356 DATA 102,4,70,5,102,4
29362 DATA 24,165,205,101,4,133
29368 DATA 205,165,206,101,5,133
29374 DATA 206,24,165,205,101,88
29380 DATA 133,205,165,206,101,89
29386 DATA 133,206,173,0,6,41
29392 DATA 7,170,160,0,173,10
29398 DATA 6,208,10,189,41,6
29404 DATA 73,255,49,205,145,205
29410 DATA 96,201,1,208,8,189
29416 DATA 41,6,17,205,145,205
29422 DATA 96,189,41,6,81,205
29428 DATA 145,205,96,0,0,0
29500 DATA 128,64,32,16,8,4,2,1

5 Utilities

5

Joystick Cursor Control

Jeff Brenner

This article will show you how to gain even more control of the Atari editing system. By using a joystick rather than the control-arrow keys, you can have instant, accurate cursor control.

This BASIC program contains a small machine language routine which will be stored in memory and executed during vertical blank. The vertical blank is the period of time between the drawing of the last line of the television screen and the movement of the electron beam to the top of the screen to begin drawing the first line. During this period, the machine language routine will be at work. Since the vertical blank occurs 60 times per second, the routine will be executed 60 times per second. The routine is executed so fast that there is no noticeable delay in computer operation.

The function of the routine is to change the joystick values into the control-arrow key codes and then store this new value in the register which the Atari uses to store keyboard data (764). Try this:

POKE 764,0

Because zero is the keyboard code (not ASCII, but an internal code) for the L character, the letter L will be displayed on the screen. The keyboard codes for the four direction keys and the corresponding joystick values follow:

CONTROL-Up = 142

CONTROL-Down = 143

CONTROL-Left = 134

CONTROL-Right = 135

Joystick Up = 14

Joystick Down = 13

Joystick Left = 11

Joystick Right = 7

Basically, here is how the program will work. Every 1/60th of a second, the routine will check the joystick port. If the joystick has been moved up, down, left, or right, then a direction code, corresponding to the position of the joystick, is stored in location 764. The Atari will then automatically display its character for that

code. In addition, a counter will be used to determine when a direction should be repeated. If the joystick is held to one position for several seconds, that direction will repeat just the way it would on a keyboard. If the joystick trigger is held down as well, the direction will repeat extra fast. Thus the joystick merely replaces the control and direction keys, and is best suited for use as a programming aid.

Joystick Cursor Control

```

5 REM JOYSTICK/CURSOR CONTROL
10 DATA 104,162,6,160,147,169,7,32,92,228,16
   9,0,133,204,133,205,133,206,96,173,120,2,
   201,15,240,24
20 DATA 197,205,240,48,133,205,201,14,240,23
   ,201,13,240,23,201,11,240,23,201,7,240,23
   ,208,6,169,0
30 DATA 133,204,133,205,76,98,228,169,142,20
   8,10,169,143,208,6,169,134,208,2,169,135,
   141,252,2,208,234
40 DATA 166,204,240,9,166,206,240,13,198,206
   ,76,98,228,169,40,133,206,133,204,208,213
   ,162,5,134,206,174
50 DATA 132,2,208,180,162,1,134,206,208,174,
   0,-1
60 I=0:C=0:RESTORE 10
70 READ N:C=C+N:IF N=-1 THEN 90
80 POKE 1664+I,N:I=I+1:GOTO 70
90 IF C=15702 THEN A=USR(1664):STOP
100 PRINT "THERE IS AN ERROR IN THE DATA"

```

Atari Verify

Michael J. Barkan

Using less than 1K of memory, this utility program for cassette can save you a lot of time and frustration.

I recently made a CSAVE and a LIST "C:" (after about five hours of typing) and *neither* of them saved the program. This sort of thing is more than distressing. My solution is neither elaborate nor entirely original, but it works.

Ed Stewart's article in *COMPUTE!'s Second Book of Atari* on backing up machine language tapes served as the inspiration for my program. Stewart's program reads a block of data from the cassette tape, puts it in a string, reads another block, adds it to the string, and so on. The string eventually contains the entire program. Of course, the string needs to be as big as the computer's memory, so I couldn't use the method directly.

I know absolutely nothing about machine language except that when I try to change something, the system crashes—so I didn't change anything. The trick was to fool the machine language program. Locations 203 and 204 (decimal) contain the starting address of string A\$. All I had to do (sounds easy, now) was reset these locations so that the machine language subroutine would "forget" that it had already put something into A\$. This means that A\$ needs to hold a maximum of only 128 bytes, the size of one cassette data block. Therefore, this program, once running, takes up less than 1K of memory; A\$ just keeps reusing the same 128 bytes.

To use this utility, type it in and save it with LIST "C:". Load the program you want to save, or start typing in a new program. Make sure your program starts at line 10 or higher. CSAVE it. Now ENTER "C:" this utility and run it. It will ask you to start loading the tape with your new program. If the tape runs all the way through and ends with an end-of-file flag, you'll get a "GOOD TAPE" message. If the tape is not readable, you'll get an error message (my favorite is 143), but *your program is still in the computer*, so you can try again. Delete lines 0 through 9 first, though.

If your tape is of the ENTER "C:" variety, just change the 255 in line 4 to 0, and the program will verify it, too.

That's all there is to it. Not quite like having a disk drive, but at least now tape storage will be far less likely to cause you distress.

Atari Verify

```

0 REM ATARI CASSETTE VERIFY UTILITY
  {9 SPACES}BY MICHAEL J. BARKAN
1 CLR :DIM A$(128):POKE 203,ADR(A$)-(INT(ADR
  (A$)/256)*256):POKE 204,INT(ADR(A$)/256):R
  EM POKE START LOCATION OF A$
2 FOR I=1536 TO 1565:READ A:POKE I,A:NEXT I:
  TRAP 7:REM POKE IN M.L. ROUTINE AND SET TR
  AP FOR END OF FILE FLAG
3 ? CHR$(125);"INSERT TAPE TO TEST":? "PRESS
  ANY KEY TO BEGIN"
4 CLOSE #1:OPEN #1,4,255,"C:":REM CHANGE 255
  TO 0 FOR TAPES WITH LONG INTER-RECORD GAP
  S
5 FOR I=1 TO 1000000:GET #1,B:X=USR(1536):REM
  LOOP THROUGH THIS MORE TIMES THAN ANYONE
  WILL EVER NEED
6 POKE 203,ADR(A$)-(INT(ADR(A$)/256)*256):PO
  KE 204,INT(ADR(A$)/256):NEXT I:REM EUREKA!
  RESET POINTER TO START OF A$
7 IF PEEK(195)=136 THEN CLOSE #1:? CHR$(125)
  ;"GOOD TAPE":END :REM LOOK FOR END OF FILE
  FLAG
8 ? "ERROR - ";PEEK(195):END :REM TAPE IS NO
  T READABLE
9 DATA 104,174,138,2,134,61,160,0,162,0,185,
  0,4,129,203,200,230,203,208,2,230,204,196,
  61,240,3,76,10,6,96

```

Automate Your Atari

Joseph J. Wrobel

Make your programs RUN automatically or PRINT a personalized message when your disk drive boots up. This short program allows you to create an AUTORUN.SYS file that will execute the commands you enter. It's easy and simple to use.

The Atari Disk Operating System (DOS) supports the use of a file named AUTORUN.SYS that has a very special characteristic. At system start-up, the DOS loads and runs this file automatically if it exists on the mounted diskette. This allows you to arrange for your Atari to come up smart.

The Potential

The AUTORUN.SYS file could contain a machine language program that loads and runs. It could also contain just a short program to do some routine operations like setting the screen margins or color before passing control to BASIC. However, the major use I've seen for AUTORUN.SYS is to direct the system to load and run a BASIC program. Not only does this type of operation save you some time and effort, but it also allows an unskilled operator, like a student, to turn on the machine and interact with an application program without getting into the details of LOAD or RUN instructions.

The Problem

So far, so good. Why doesn't everyone use the AUTORUN.SYS file? Apparently the major obstacle to its more widespread use is the fact that it is a machine language routine. Thus, it requires knowledge of 6502 machine language and, for complex operations, some knowledge of the intricacies of the Atari Operating System to create a functional AUTORUN.SYS file. Unless someone came up with a program to do it for you.

"Automate" (Program 1) is just such a program. If you key in this program correctly and run it, Automate will help you create your own personal AUTORUN.SYS file, and it won't hurt a bit. The program starts by asking you to input the series of commands you wish to be executed at start-up. You enter the commands

exactly as you would if the machine came up in its normal ready state. The only limit on the number of commands is that the total number of characters entered may not exceed 196 (including the Atari end-of-line character added each time you hit RETURN). The program keeps track of the number of characters entered and will prevent you from exceeding this limit. After you've entered the final command in the sequence, the program will create an AUTORUN.SYS file on the mounted diskette. Note that any previous AUTORUN.SYS file will be overwritten by this operation.

The next time you boot up from the diskette bearing the AUTORUN.SYS file, the AUTORUN.SYS program will be run. This will cause the commands you entered to be executed in the order they were entered (although they will not be displayed), then control will be returned to the system. The commands, of course, must be compatible with the cartridge in use (BASIC, Assembler Editor, etc.) or an error will result. If at any time you wish to boot up from a diskette and circumvent the AUTORUN.SYS file, just hold the OPTION key down until system initialization is complete. The AUTORUN.SYS file created by Automate checks that key and, if it finds it depressed, the command list will not be executed.

A BASIC Example

To demonstrate the use of the program, a single command BASIC example will be presented. Let us suppose there exists a BASIC program entitled BEGIN which you would like to run automatically at start-up. Using Automate, you enter (as Command #) the statement:

GR.0:?"Autoboot in progress.":RUN"D:BEGIN"

then press RETURN. Assuming you entered the command correctly, you respond to the question:

Is that correct (Y/N)?

by pressing Y. When the program asks if there are:

More commands (Y/N)?

respond by pressing N. The program then creates the AUTORUN.SYS file and displays READY when it's done. If you now turn off your computer and switch it on again, you will find that it "comes up" running program BEGIN. How simple can you get?

Description of Operation

This section is for those who are not satisfied with just running the program, but are also interested in knowing how it works. Let's first take another look at Program 1. Automate consists of three major sections. The first section (lines 50 through 130) are for documentation and initialization. The program employs two key numeric variables: I, which counts the number of commands entered, and L, which counts the total number of characters in the command list. The second program section (lines 140 through 350) INPUTs the commands one at a time. As each command is entered, the program allows for error correction, checks command list size, packs the command into B\$ and tacks on an ATARI end-of-line (EOL) character, namely CHR\$(155). The third section of the program (lines 360 through 600) actually creates the AUTORUN.SYS file.

Before this third section is discussed, I direct your attention to Program 2. This is the assembly listing for the core of the AUTORUN.SYS program. What this machine language program does, in a nutshell, is to temporarily take over the task of supplying screen editor data by substituting a new device handler table and "get character" routine for the default ones provided by the operating system. At system start-up while the AUTORUN.SYS program is active, it intercepts all the keyboard entry requests and feeds out, one character at a time, the commands which you have entered. When it has sent out the last character of the last command in the list, it re-installs the default screen editor handler table, and the system takes over from there.

Returning to the section of the BASIC program which creates the AUTORUN.SYS file, you will find that it consists primarily of three loops. Loop one (lines 490 through 510) PUTs the core program and its associated 6-byte header into the file as READ from the DATA statements in lines 430 through 480.

Note that in line 500 of Automate, two numbers are changed from the values shown in the DATA statements before putting them into the AUTORUN.SYS file. The first is a byte in the AUTORUN.SYS file header which gives the end of the program when loaded in memory. This is the sum of the core program length and the number of bytes in the command list. Automate also alters the value of the immediate argument of the CPY instruction in line 370 of Program 2. This byte is set equal to the total number of characters (including EOLs) in the command list.

Loop two (lines 530 through 550) PUTs in the command list which resides in B\$. Finally, loop three (lines 580 through 590) adds a 12-byte postscript to the file, which provides the system with the initialization and run locations for the routine.

The BASIC program here provides an easy way to create a useful AUTORUN.SYS file. There are dozens of ways this file can be used. It doesn't necessarily have to be a serious application. For example, it's sort of fun just to start up my machine, listen to it go through its disk machinations, then see it automatically display the personalized greeting:

READY WHEN YOU ARE, J.W.!

Program 1. Automate

```

50 I=0:L=0:MAX=196
60 DIM A$(MAX),B$(MAX),R$(1)
70 OPEN #1,4,0,"E:":OPEN #2,4,0,"K:"
80 ? "This program helps you to create"
90 ? " a personalized AUTORUN.SYS file"
100 ? " which, following the disk boot"
110 ? "{3 SPACES}process, automatically issu
    es"
120 ? "{4 SPACES}a set of commands that plot"
130 ? "{5 SPACES}specify."
140 I=I+1
150 ? :? "Please enter command #";I;". "
160 ? :INPUT #1;A$
170 POKE 766,1:?? "Command #";I;": ";A$:POKE
    E 766,0
180 ? :? "Is that correct (Y/N)? ";:GET #2,X
    :? :R$=CHR$(X)
190 IF R$="Y" OR R$="y" THEN 220
200 IF R$="N" OR R$="n" THEN 150
210 GOTO 170
220 X=L+LEN(A$)+1-MAX
230 IF X<0 THEN 260
240 ? :? "Command #";I;" is ";X;" character (
    s)"
250 ? "too long.":I=I-1:GOTO 270
260 B$(L+1)=A$:L=LEN(B$):B$(L+1)=CHR$(155):L
    =L+1
270 ? :? "Current command list:"
280 POKE 766,1:?? B$:POKE 766,0
290 IF L>=MAX-1 THEN ? "Command list is full
    .":? :GOTO 370
300 ? "Command list can hold ";MAX-L-1;" mor
    e"

```

```

310 ? " character(s)."
320 ? :? "More commands (Y/N)? " ;:GET #2,X:R
    $=CHR$(X)
330 IF R$="Y" OR R$="y" THEN 140
340 IF R$="N" OR R$="n" THEN 360
350 GOTO 300
360 ? CHR$(125);
370 ? "Mount diskette which is to bear"
380 ? " the AUTORUN.SYS file, then"
390 ? " press RETURN. " ;:GET #2,X:CLOSE #1:
    CLOSE #2
400 ? CHR$(125);: ? "Writing AUTORUN.SYS file
    ."
410 OPEN #1,8,0,"D:AUTORUN.SYS"
420 REM PUT OUT THE HEADER AND THE CORE MACH
    INE LANGUAGE PROGRAM
430 DATA 255,255,0,6,59,6
440 DATA 173,31,208,41,4,240,10,169,18,141,3
    3,3
450 DATA 169,6,141,34,3,96,251,243,51,246,33
    ,6
460 DATA 163,246,51,246,60,246,76,228,243,0,
    238,33
470 DATA 6,172,33,6,192,0,208,10,169,0,141,3
    3
480 DATA 3,169,228,141,34,3,185,59,6,160,1,9
    6
490 FOR I=1 TO 66:READ X
500 IF I=5 OR I=48 THEN X=X+L
510 PUT #1,X:NEXT I
520 REM ADD THE COMMAND LIST
530 FOR I=1 TO L
540 X=ASC(B$(I,I))
550 PUT #1,X:NEXT I
560 REM APPEND INITIALIZE AND RUN VECTORS
570 DATA 226,2,227,2,0,6,224,2,225,2,17,6
580 FOR I=1 TO 12:READ X
590 PUT #1,X:NEXT I
600 CLOSE #1: ? CHR$(125);:END

```

Program 2. Assembly Listing

```

D01F          0100  CONSOL  =      $D01F
0320          0110  DEVTAB  =      $0320
E400          0120  OLDDHT  =      $E400
              0130  ;
0000          0140          *=      $0600
0600  AD1FD0  0150  INIT    LDA  CONSOL
              ;Load the console switch register

```

```

0603 2904    0160          AND    #$04
        ;and check for the OPTION key.
0605 F00A    0170          BEQ    RUN
        ;If it's pressed, branch to the RTS.
0607 A912    0180          LDA    #NEWDHT&#00FF
        ;Otherwise, install the vector
0609 8D2103  0190          STA    DEVTAB+1
        ;to the new device handler table
060C A906    0200          LDA    #NEWDHT/256
        ;in the appropriate place in the
060E 8D2203  0210          STA    DEVTAB+2
        ;device table and
0611 60      0220 RUN      RTS
        ;return.
        0230 ;
0612 FBF3    0240 NEWDHT .WORD $F3FB
        ;This is the replacement
0614 33F6    0250          .WORD $F633
        ;screen editor handler
0616 2106    0260          .WORD GET-1
        ;vector table. All the
0618 A3F6    0270          .WORD $F6A3
        ;vectors have their default
061A 33F6    0280          .WORD $F633
        ;values except for the
061C 3CF6    0290          .WORD $F63C
        ;GET routine, which
061E 4C      0300          .BYTE $4C
        ;points to the replacement
061F E4F3    0310          .WORD $F3E4
        ;routine below.
        0320 ;
0621 00      0330 COUNTR .BYTE 0
        ;character counter
        0340 ;
0622 EE2106  0350 GET      INC    COUNTR
        ;Increment the character
0625 AC2106  0360          LDY    COUNTR
        ;counter. Compare it with
0628 C000    0370          CPY    #ENDLST-BEGLST
        ;the command list length.
062A D00A    0380          BNE    CONT
        ;If not equal, branch to CONT.
062C A900    0390          LDA    #OLDDHT&#00FF
        ;Otherwise, reinstate the
062E 8D2103  0400          STA    DEVTAB+1
        ;default screen editor handler
0631 A9E4    0410          LDA    #OLDDHT/256
        ;table vector at the proper

```

```
0633 8D2203 0420          STA  DEVTAB+2
      ;spot in the device table.
0636 B93B06 0430 CONT     LDA  BEGLST-1,Y
      ;Fetch the next character
0639 A001 0440          LDY  #1
      ;from the command list and
063B 60 0450          RTS
      ;return.
      0460 ;
      0470 BEGLST
      0480 ;The command list goes here
063C      0490 ENDLST .END
```


The Wedge: Adding Commands To Atari BASIC

Charles Brannon

You can customize your Atari BASIC by adding new commands to the language itself. To demonstrate how to do it, the program below adds five DOS commands to BASIC—including a directory command. There are two versions of the same program. Program 1 is a BASIC loader. Type it in normally, and it creates a machine language program for you from the information in the DATA statements. Program 2 is an assembly listing of the same routine. It shows how the machine language works and is useful to programmers who know machine language or want to learn more about it. It's not necessary, however, to understand Program 2 in order to make good use of Program 1.

A letter published some months ago in *COMPUTE!'s* "Ask The Readers" column regretted the need for "this POKE or that POKE" to accomplish various tasks. The required solution is an "expanded command set." An enticing prospect, adding commands to a language, and a seemingly impossible one, too.

Atari BASIC, like most microcomputer BASICs, is burned into nonvolatile ROM memory. The machine language routines to list, save, edit, and run your program cannot be altered or patched in any way. (However, on a 48K Atari, you can copy the BASIC cartridge to disk as a binary file, modify it with a machine language monitor, and load it into the top of memory where it will act almost as a ROM cartridge.)

The most common (and easiest to implement) extension of a language is the addition of immediate mode commands. These direct commands, which are not usually executed in a program, but from the keyboard, include RUN, SAVE, LIST, NEW, DOS, etc. Thanks to Atari's modular Operating System (OS), we can easily add this type of command.

An Overview of Atari's Operating System

To understand how the wedge works, we'll have to delve into the

mysterious 10K ROM. If you just want to use the program and aren't concerned about the technical details, feel free to skip ahead. The operating system (OS) of a computer is responsible for all input and output to and from disk, cassette, printer, and keyboard. It can also perform such chores as memory management and screen display. On many microcomputers, the OS does not exist as a separate entity, but is incorporated into the BASIC interpreter.

The Atari, on the other hand, is the first microcomputer with a general-purpose, plug-in operating system. This goes hand in hand with the use of program and game cartridges. All programs running on an Atari use a common set of routines, from floating point arithmetic to high-resolution graphics routines such as PLOT, DRAWTO, and FILL.

A Mini-Language

So, instead of BASIC providing a marginal operating system (which on many machines is a maze of machine language calls, requiring incompatible register setup and initialization), we have a BASIC cartridge which uses universal OS routines. A good OS simulates a mini-language. It provides documented, unchanging (between various revisions), unified subroutines with full parameter passing and error-checking.

Furthermore, a good OS is *extensible*. All the major routines and subroutines are accessed *indirectly*, through pointers. That is why the Atari is so flexible. If you want to change the personality of your computer, just change one of the *vectors* of a given routine to point to your machine language routine. Your program can then pass on control to the default program.

A Flexible Computer

This indirection is visible throughout the Atari. At the low end is color indirection, where you can change the color of anything drawn to another color merely by changing one *color register*. The default character set pointer can be changed to point to a user-designed character set. The system interrupt routines and display list interrupts are all fully accessible via a table of pointers. The BREAK key can be masked; the keyboard scan routine can be modified or by-passed; exotic peripherals can be serviced. And *all* input/output devices are user-definable, from the keyboard to the disk drive.

A notable peculiarity of the Atari is that not just the disk

drive or printer, but also the TV screen and keyboard, are considered peripherals. You don't print a character to the screen on the Atari; you send a character or buffer to the Editor device.

Chain of Command

Through the hierarchy of a subset of the OS, the CIO (Central Input/Output), BASIC politely requests a line of input from screen and keyboard. After BASIC makes this request, control is passed to CIO, which calls the Editor. The Editor lets the user enter a line of text (which can be up to three screen lines long). The user can use cursor controls to edit the line or to move the cursor anywhere on the screen to edit another line.

When RETURN is pressed, the line the cursor is on is placed into a buffer (block of memory). Next, CIO gives this information to the calling routine via another buffer. The CIO is designed to be easy to use from machine language. If you think it sounds complicated, imagine performing all these tasks *without* an operating system.

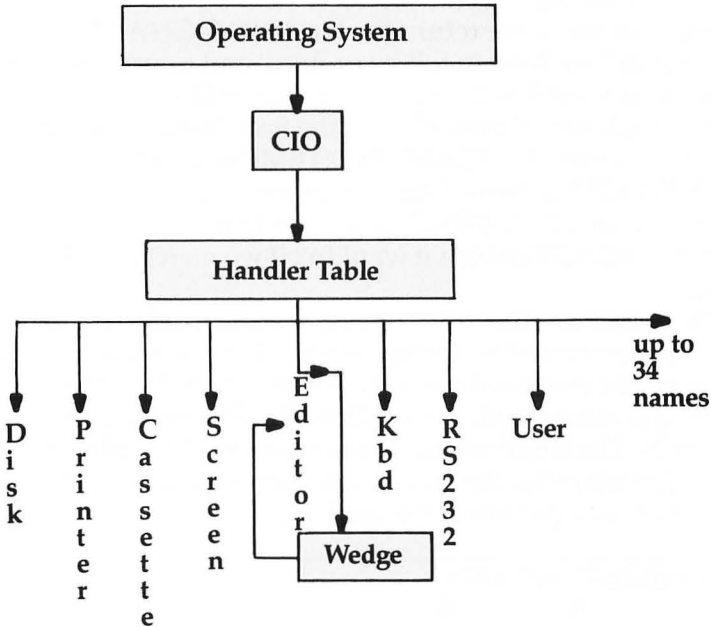
Driving a Wedge

We don't have to modify BASIC at all. We just "wedge" our way into the Editor device E:. As intimated, even the "system" devices such as E: or D: (the disk driver) can be replaced. Usually, however, you don't want to replace a vectored routine; you just want to insert an additional task. In this case, you point the vector to your routine, which performs the little extra task and then calls the main routine. This by-pass explains the term *wedge*.

The Handler Table contains the names of all the devices. If you wanted to, you could change the name of the cassette device (C:) to another character, such as T: (for tape), by finding the C in the table and changing it to a T. Along with each name, the Handler Table includes an address that points to another table of addresses that point to all the functions of that particular device. This is multilevel indirection. There is even a vector that points to a list of vectors!

We want to modify the Editor, so we change the first vector to point to *our* list of vectors. All we really need to do is change one of the vectors in the Editor's list of vectors, the "Get Character" address. Since this list is in ROM, at \$E400, we need to copy this 16-byte table to RAM, modify it, and repoint the Handler Table to our RAM version of the Editor Handler Table.

Wedging into a Vector



A Monitor Monarchy

Now that we've got the operating system calling our routine instead of the Editor in ROM, we've got total control of almost all console input/output. The Get Character routine, instead of calling E:, asks us for an ASCII character, presumably from the screen and keyboard. We comply by calling the default routine in ROM.

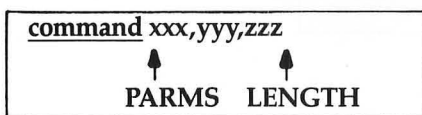
This seems rather roundabout, doesn't it? But we reserve the right to monitor all characters returned to the operating system, and hence, BASIC. We get to examine every line of input before that line is returned to BASIC, where any strange new commands would be scorned with an error message.

So, we just catch the carriage return code and leisurely examine the input buffer, located at \$0580. All we have to do is compare it against a table of commands, and, if we find a match, execute the command. If not, we just return the line to CIO (and CIO gives it back to BASIC) on the assumption that it's either a blank line, a BASIC command, or a syntax error. Sounds simple, but such a "parsing" routine is quite a headache to code and understand.

A REMarkable Solution

After we've intercepted and executed the line, how do we prevent a syntax error when we return the line to BASIC? (And since we've "cut in," we have to follow protocol and return *something*.) One solution would be to erase the buffer by filling it with spaces. An easier trick would be to change the first character of the line to a period; for example, SCRATCH D:TEMP would become .CRATCH D:TEMP. Since BASIC interprets a leading period as an abbreviation for REM, BASIC conveniently ignores the command and returns READY (which it wouldn't if we merely blanked out the line).

The parser routine makes it easy for you to add commands. Just place the name of each command, followed by a zero, and the address where you want control to be transferred after the command is recognized, in COMTBL (COMmand TaBLe, see Program 2). The length of the line is found in LENGTH, and the second character after the command is returned in PARMS (since this is where any parameters would be).



Note that the length is one character past the end of the string, assuming you number from zero. Your command processor can find the command string in LBUFF (\$0580).

Theoretically, this technique can be used to add commands to *any* language environment. You only have to find a way to make the language processor ignore commands when you return the line (such as blanking it out). Of course, the commands themselves are usually language-specific.

Copious Commands

Now the way is open to add a plethora of BASIC utility commands. Of course, these will have to be written in machine language and interfaced with the Wedge. I've included the resident DOS commands LOCK, UNLOCK, RENAME, and SCRATCH, as well as DIR to print the directory.

You can study the assembly listing (Program 2). If you have an assembler, try typing it in and modifying it. It contains a wealth of techniques and information, such as pattern matching,

indirect subroutine calls, making a routine "RESET-proof," using CIO for input/output from machine language, long branching, modular programming, calling BASIC's ERROR routine, even pressing SYSTEM RESET from within a program.

Using the Wedge

A machine language program can be hard to enter into the Atari without an assembler. Program 1 will write the machine language to disk in the form of an AUTORUN. SYS file. Save this program so you can write copies to any disk. When you boot this disk the AUTORUN file will automatically load and initialize the Wedge. You can use the Wedge's console DOS directly, without waiting for the disk utility package (DUP.SYS) to load in, and without losing any programs in memory.

Commands provided are DIR (lists the directory of drive one), LOCK, UNLOCK, SCRATCH (delete), and RENAME. Remember to include the D: (or D2: for drive two, if you have one) in the filename with all the commands except DIR. With RENAME, use the convention RENAME D: oldname, newname.

The Wedge is "persistent"; in other words, it reinitializes itself when you press SYSTEM RESET, so it's kind of hard to get rid of it. An additional command, KILL, removes the Wedge. You can bring back the Wedge with PRINT USR (7936).

These commands are just a start. Many others are possible: RENUMBER, FIND, AUTO line number, UPDATE (removes unused variables from the variable name table), and more.

Program 1. BASIC Wedgemaker

```

100 REM WEDGE BASIC LOADER
110 GRAPHICS 0: ? "Insert a DOS 2.05 diskette
   "
120 ? "with DOS.SYS in drive 1."
130 ? : ? "Press RETURN when you have done th
   is."
140 IF PEEK(764)<>12 THEN 140
150 POKE 764,255
160 ? : ? "Now writing the Wedge AUTORUN.SYS
   file"
170 TRAP 190
180 OPEN #1,8,0,"D:AUTORUN.SYS":TRAP 40000:G
   OTO 200
190 CLOSE #1: ? : ? "Can't open AUTORUN.SYS fo
   r write.":END
200 PUT #1,255:PUT #1,255:REM $FFFF HEADER

```

```

210 PUT #1,0:PUT #1,31:REM $1F00 START
220 PUT #1,74:PUT #1,33:REM $214A END
230 FOR I=7936 TO 8522+6:REM INCLUDE 6-BYTE
    AUTORUN
240 READ A:TRAP 310:PUT #1,A:TRAP 40000
250 CKSUM=CKSUM+A
260 NEXT I
270 IF CKSUM<>60435 THEN ? "{BELL}Bad number
    in DATA statements.":ERR=1
280 CLOSE #1
290 IF NOT ERR THEN ? :? "DATA ok, write su
    ccessful."
300 END
310 ? :? "Error-";PEEK(195);" when attemptin
    g disk write.":CLOSE #1:END
320 REM
330 REM Following is the decimal
340 REM equivalent of Wedge 1.0
350 REM Must be typed in perfectly
360 REM in order to function.
370 REM
7936 DATA 104,165,12,141,37,31
7942 DATA 165,13,141,38,31,169
7948 DATA 36,133,12,169,31,133
7954 DATA 13,32,43,31,32,92
7960 DATA 31,169,75,141,231,2
7966 DATA 169,33,141,232,2,96
7972 DATA 32,64,21,32,11,31
7978 DATA 96,169,80,141,68,3
7984 DATA 169,31,141,69,3,169
7990 DATA 0,141,73,3,169,12
7996 DATA 141,72,3,169,11,141
8002 DATA 66,3,162,0,32,86
8008 DATA 228,152,48,1,96,76
8014 DATA 55,33,65,116,97,114
8020 DATA 105,32,87,101,100,103
8026 DATA 101,155,160,0,185,26
8032 DATA 3,201,69,240,7,200
8038 DATA 200,192,34,208,243,96
8044 DATA 200,169,165,153,26,3
8050 DATA 200,169,31,153,26,3
8056 DATA 162,0,189,0,228,157
8062 DATA 165,31,232,224,16,208
8068 DATA 245,169,184,141,169,31
8074 DATA 169,31,141,170,31,24
8080 DATA 173,4,228,105,1,141
8086 DATA 186,31,173,5,228,105
8092 DATA 0,141,187,31,169,0
8098 DATA 133,203,96,251,243,51

```

8104 DATA 246,184,31,163,246,51
8110 DATA 246,60,246,76,228,243
8116 DATA 56,1,1,125,32,32
8122 DATA 62,246,8,201,155,240
8128 DATA 4,230,203,40,96,140
8134 DATA 181,31,142,182,31,165
8140 DATA 203,240,86,169,51,133
8146 DATA 205,169,32,133,206,160
8152 DATA 0,177,205,217,128,5
8158 DATA 208,12,200,177,205,240
8164 DATA 40,196,203,208,240,76
8170 DATA 37,32,201,255,240,53
8176 DATA 160,0,177,205,240,9
8182 DATA 230,205,144,2,230,206
8188 DATA 76,242,31,24,165,205
8194 DATA 105,3,133,205,144,2
8200 DATA 230,206,76,215,31,200
8206 DATA 132,204,177,205,141,183
8212 DATA 31,200,177,205,141,184
8218 DATA 31,108,183,31,160,0
8224 DATA 169,46,153,128,5,169
8230 DATA 0,133,203,169,155,172
8236 DATA 181,31,174,182,31,40
8242 DATA 96,68,73,82,0,125
8248 DATA 32,83,67,82,65,84
8254 DATA 67,72,0,22,33,76
8260 DATA 79,67,75,0,27,33
8266 DATA 85,78,76,79,67,75
8272 DATA 0,32,33,82,69,78
8278 DATA 65,77,69,0,37,33
8284 DATA 75,73,76,76,0,42
8290 DATA 33,255,155,50,54,32
8296 DATA 70,82,69,69,32,83
8302 DATA 69,67,84,79,82,83
8308 DATA 155,155,0,0,68,58
8314 DATA 42,46,42,162,80,169
8320 DATA 12,157,66,3,32,86
8326 DATA 228,162,80,169,3,157
8332 DATA 66,3,169,6,157,74
8338 DATA 3,169,120,157,68,3
8344 DATA 169,32,157,69,3,32
8350 DATA 86,228,152,16,3,76
8356 DATA 55,33,162,80,169,5
8362 DATA 157,66,3,169,100,157
8368 DATA 68,3,141,68,3,169
8374 DATA 32,157,69,3,141,69
8380 DATA 3,169,20,157,72,3
8386 DATA 141,72,3,32,86,228
8392 DATA 152,48,13,169,9,141


```

8398 DATA 66,3,162,0,32,86
8404 DATA 228,76,166,32,162,80
8410 DATA 169,12,157,66,3,32
8416 DATA 86,228,76,30,32,162
8422 DATA 80,157,66,3,169,0
8428 DATA 157,73,3,164,203,153
8434 DATA 128,5,56,152,229,204
8440 DATA 157,72,3,24,169,128
8446 DATA 101,204,157,68,3,169
8452 DATA 5,105,0,157,69,3
8458 DATA 32,86,228,152,16,3
8464 DATA 76,55,33,76,30,32
8470 DATA 169,33,76,229,32,169
8476 DATA 35,76,229,32,169,36
8482 DATA 76,229,32,169,32,76
8488 DATA 229,32,173,37,31,133
8494 DATA 12,173,38,31,133,13
8500 DATA 76,116,228,72,162,80
8506 DATA 169,12,157,66,3,32
8512 DATA 86,228,104,162,255,154
8518 DATA 133,185,76,64,185
9000 REM DATA FOR AUTORUN ADDRESS
9010 DATA 224,2,225,2,1,31
9020 REM END OF DATA STATEMENTS

```

Program 2. Wedge Assembly Source

```

0100 ; The Atari Wedge
0110 ;
0120          *=$1F00
0130 ICCOM      =$0342
0140 ICBADR     =$0344
0150 ICBLN      =$0348
0160 ICAUX1     =$034A
0170 COPN       =$03
0180 CPTXTR     =$09
0190 CGTXTR     =$05
0200 CPBINR     =$0B
0210 CCLOSE     =$0C
0220 CIO        =$E456
0230 OPDIR      =$06
0240 HATABS     =$031A
0250 LBUFF      =$0580
0260 LENGTH     =$CB
0270 MEMLO      =$02E7
0280 PARMS      =$CC
0290 COM         =$CD
0300 DOSINIT     =$0C
0310 ENTRY      PLA          ;For BASIC ini
                                tialization

```

```

0320 ; Make wedge "RESET-proof"
0330 INIT
0340          LDA DOSINIT          ; Save DOS
0350          STA REINIT+1        ; initializatio
n
0360          LDA DOSINIT+1        ; inside the RE
INIT
0370          STA REINIT+2        ; JSR call
0380 ;
0390 INIT2    LDA #REINIT&255     ; Replace DOS i
nit
0400          STA DOSINIT          ; with Wedge
0410          LDA #REINIT/256      ; init
0420          STA DOSINIT+1
0430          JSR MSG ; Print message
0440          JSR ECHANGE          ; hookup new E
:
0450          LDA #ENDWEDGE&255    ; Bump up
0460          STA MEMLO
0470          LDA #ENDWEDGE/256    ; low memory
pointer
0480          STA MEMLO+1
0490          RTS
0500 ;
0510 REINIT   JSR XXXX             ; XXXX is filled
in with DOSINIT
0520          JSR INIT2
0530 XXXX     RTS
0540 ;
0550 ; Print "welcome" message
0560 ;
0570 MSG      LDA #WMSG&255       ; Store address
of
0580          STA ICBADR           ; message
0590          LDA #WMSG/256
0600          STA ICBADR+1
0610          LDA #0              ; Set length
0620          STA ICBLN+1
0630          LDA #12
0640          STA ICBLN
0650          LDA #CPBINR         ; Ignore carriag
e-returns
0660          STA ICCOM
0670          LDX #0              ; File 0, the ed
itor
0680          JSR CIO             ; Call CIO to pr
int it
0690          TYA
0700          BMI ERR            ; If no error, r
eturn

```

```

0710          RTS
0720 ERR      JMP ERROR
0730 ;
0740 WMSG      .BYTE "Atari Wedge 2.0",155
0750 ;
0760 ; Following replaces the old E:
0770 ;
0780 EXCHANGE  LDY #0          ;Search for E:
0790 ELOOP     LDA HATABS,Y    ;in handler tab
le
0800          CMP #'E
0810          BEQ EFOUND      ;Found end?
0820          INY              ;no, next entry
0830          INY
0840          CPY #34          ;end of table?
0850          BNE ELOOP
0860          RTS              ;return
0870 ;
0880 ; Store new handler table address
0890 ;
0900 EFOUND     INY
0910          LDA #WEDGETAB&255
0920          STA HATABS,Y
0930          INY
0940          LDA #WEDGETAB/256
0950          STA HATABS,Y
0960 ; Transfer Editor table to Wedge table
0970          LDX #0
0980 XFER       LDA $E400,X
0990          STA WEDGETAB,X
1000          INX
1010          CPX #16
1020          BNE XFER
1030 ; Patch in MYINPUT routine
1040          LDA #MYINPUT-1&255
1050          STA WEDGETAB+4
1060          LDA #MYINPUT-1/256
1070          STA WEDGETAB+5
1080          CLC
1090          LDA $E404          ; Get character
address
1100          ADC #1              ; Actual address
s is +1
1110          STA MYINPUT+1      ; Egads!
1120          LDA $E405          ; Self-modifyin
g code!
1130          ADC #0              ; (Accept any ca
rry)
1140          STA MYINPUT+2

```

```

1150          LDA #0
1160          STA LENGTH      ;Clear length i
           initially
1170          RTS
1180 ;
1190 ; Wedge handler address table
1200 WEDGETAB *=*+16
1210 YSAVE   *=*+1          ;Used to save Y
           register
1220 XSAVE   *=*+1          ;Ditto for X
1230 JUMPADR *=*+2          ;used for indir
           ect JMP
1240 MYINPUT
1250 ; The $F63E address is actually placed
           here by above code
1260 ; to permit this routine to run on the
           Revision B OS
1270 ; (where it wouldn't necessarily be $F6
           3E)
1280          JSR $F63E      ;Get a character
           from E:
1290          PHP
1300          CMP #155        ;End of line? (
           CR)
1310          BEQ ENDLINE    ;Yes, complete
           line ready
1320          INC LENGTH
1330          PLP
1340          RTS            ;No, let CIO ha
           ve the character
1350 ENDLINE
1360          STY YSAVE        ;Save Y for CIO
1370          STX XSAVE
1380          LDA LENGTH
1390          BEQ RETURN.LINE
1400 LOOKUP
1410          LDA #COMTBL&255 ;Set up indire
           ct pointer for
1420          STA COM
1430          LDA #COMTBL/256 ;command table
1440          STA COM+1
1450 NEXTCOM  LDY #0
1460 COMPL0OP
1470          LDA (COM),Y      ;Compare comman
           d against line buffer
1480          CMP LBUFF,Y      ;Okay so far?
1490          BNE NOTSAME      ;no match
1500          INY
1510          LDA (COM),Y      ;is next charac
           ter null?

```

```

1520      BEQ COMFOUND      ;yes, command f
      ound
1530      CPY LENGTH      ;exceeded limit
      s?
1540      BNE COMLOOP      ;if not, contin
ue comparision
1550      JMP RETURN.LINE ;give line to
      language
1560 NOTSAME  CMP #255      ;End of table?
1570      BEQ RETURN.LINE
1580      LDY #0           ;No, skip over
      command
1590 FINDEND  LDA (COM),Y
1600      BEQ ENDCOM       ;Hit the zero y
      et?
1610      INC COM         ;No, next chara
      cter
1620      BCC NOINC1
1630      INC COM+1
1640 NOINC1   JMP FINDEND  ;continue until
      null byte found
1650 ENDCOM   CLC          ;Add 3 to skip o
      ver null byte
1660      LDA COM         ;and JMP address
1670      ADC #3
1680      STA COM
1690      BCC NOINC2      ;Check for carry
1700      INC COM+1
1710 NOINC2   JMP NEXTCOM
1720 COMFOUND
1730      INY
1740      STY PARMS       ;Y is index into
      parameters
1750      LDA (COM),Y      ;Load JUMPADR w
      ith command address
1760      STA JUMPADR
1770      INY
1780      LDA (COM),Y
1790      STA JUMPADR+1
1800      JMP (JUMPADR)    ;Execute!
1810 EXIT     LDY #0       ;Commands retur
      n here
1820      LDA #'.'         ;Change first c
      haracter to
1830      STA LBUF,Y       ;".", or REM
1840      ;               Allows BASIC to
      ignore line
1850 RETURN.LINE
1860      LDA #0

```

```

1870          STA LENGTH
1880 NOAUTO
1890          LDA #155          ;Return EOL to
      CIO
1900          LDY YSAVE        ;Restore Y
1910          LDX XSAVE        ;and X
1920          PLP              ;and processor

      status
1930          RTS              ;That's it
1940 COMTBL
1950 ; Wedge commands and command table
1960 ; Format is:
1970 ; .BYTE "COMMAND",0
1980 ; .WORD COMMAND.ADDRESS
1990 ; End of table is
2000 ; .BYTE 255
2010          .BYTE "DIR",0
2020          .WORD DIR
2030          .BYTE "SCRATCH",0
2040          .WORD SCRATCH
2050          .BYTE "LOCK",0
2060          .WORD LOCK
2070          .BYTE "UNLOCK",0
2080          .WORD UNLOCK
2090          .BYTE "RENAME",0
2100          .WORD RENAME
2110          .BYTE "KILL",0
2120          .WORD KILL
2130          .BYTE 255
2140 ;
2150 DIRBUF      *=*+20
2160 DIRNAME     .BYTE "D:*. *"
2170 ;
2180 ; Start of commands:
2190 ;
2200 DIR
2210          LDX #$50          ; IOCB#5
2220          LDA #CCLOSE
2230          STA ICCOM,X
2240          JSR CIO          ;CLOSE#5
2250 ; OPEN#5,6,0,"D:*. *"
2260          LDX #$50          ;channel#5
2270          LDA #COPN        ;open command
2280          STA ICCOM,X
2290          LDA #OPDIR        ;special "direc
      tory" command
2300          STA ICAUX1,X
2310          LDA #DIRNAME&255 ;filename (wi
      ldcard)

```

```

2320          STA ICBADR,X
2330          LDA #DIRNAME/256
2340          STA ICBADR+1,X
2350          JSR CIO          ;set it up!
2360          TYA
2370          BPL NOERR1
2380          JMP ERROR
2390 ; Print a line to the Editor
2400 NOERR1
2410 NEXT      LDX #$50          ;#5
2420          LDA #CGTXTR        ;Get a line
2430          STA ICCOM,X
2440          LDA #DIRBUF&255 ;Put it into t
           he buffer
2450          STA ICBADR,X
2460          STA ICBADR
2470          LDA #DIRBUF/256
2480          STA ICBADR+1,X
2490          STA ICBADR+1
2500          LDA #20              ;Maximum leng
           th is 20
2510          STA ICBLEN,X        ; (actually 17)
2520          STA ICBLEN
2530          JSR CIO
2540          TYA                  ;Check for end
           of file
2550          BMI ENDIR          ;On error, fin
           ished directory
2560 NOERR2    LDA #CPTXTR        ;Put text reco
           rd (print a line)
2570          STA ICCOM
2580          LDX #0              ;Channel 0 is
           open to the Editor
2590          JSR CIO
2600          JMP NEXT          ;Read next lin
           e
2610 ;
2620 ENDIR     LDX #$50          ;CLOSE#5
2630          LDA #CCLOSE
2640          STA ICCOM,X
2650          JSR CIO
2660          JMP EXIT
2670 ;End of directory routine
2680 ;
2690 ;Following routine is used by lock
2700 ;unlock, scratch, and rename
2710 ;Filename buffer is in LBUFF
2720 ;e.g. LOCK D:TEMP
2730 ; this ^ portion is used

```

```

2740 ; to tell CIO the filename.
2750 CALLCIO
2760      LDX ##50      ; Use file 5 (XIO
      n, #5, etc.)
2770      STA ICCOM, X   ; Store command
2780      LDA #0         ; Clear MSB
2790      STA ICBLN+1, X ; of length
2800      LDY LENGTH
2810      STA LBUFF, Y
2820      SEC            ; Get length
2830      TYA            ; of filename
2840      SBC PARMS      ; (skip over comm
and name)
2850      STA ICBLN, X
2860      CLC
2870      LDA #LBUFF&255 ; PARMS is start
of parameters,
2880      ADC PARMS      ; the space in LB
UFF
2890      STA ICBADR, X  ; after the comma
nd
2900      LDA #LBUFF/256
2910      ADC #0         ; Catch any carry
2920      STA ICBADR+1, X
2930      JSR CIO ; Do the job
2940      TYA
2950      BPL NOERR3
2960      JMP ERROR
2970 NOERR3  JMP EXIT
2980 ;
2990 SCRATCH LDA #33
3000      JMP CALLCIO
3010 LOCK   LDA #35
3020      JMP CALLCIO
3030 UNLOCK LDA #36
3040      JMP CALLCIO
3050 RENAME LDA #32
3060      JMP CALLCIO
3070 ;
3080 ; Remove Wedge
3090 ;
3100 KILL   LDA REINIT+1 ; Restore old DOS
3110      STA DOSINIT   ; vector
3120      LDA REINIT+2
3130      STA DOSINIT+1
3140      JMP $E474      ; "Press" SYSTEM RE
SET
3150 ;
3160 ; End of current wedge

```



```

3170 ; (Although more commands can be added.
3180 ; See future issues of COMPUTE!
3190 ;
3200 ERROR      PHA          ;Save error code
3210           LDX #$50      ;close file 5
3220           LDA #CCLOSE
3230           STA ICCOM,X
3240           JSR CIO
3250           PLA          ;retrieve error cod
e
3260           LDX #$FF      ;reset stack
3270           TXS
3280           STA $B9        ;tell BASIC the err
or code
3290           JMP $B940     ;call the ERROR rou
tine
3300 ;                      in the BASIC cartr
idge
3310 ;
3320 ENDWEDGE
3330 ; Autorun
3340 ;
3350           *=$02E0
3360           .WORD INIT
3370 ;
3380           .END

```

Renumber Plus

Manny Juan and Paul N. Havey

A renumbering utility is an important tool for the BASIC programmer. You will find "Renumber Plus" to be an invaluable aid.

When you type a BASIC statement and press RETURN, BASIC converts your code into tokens. For example, all keywords and variables become one-byte tokens. A string becomes a sequence of tokens. The first byte of the sequence—always the decimal number 15—tells BASIC that a string follows. The second byte tells BASIC the length of the string in bytes. The string appears as ASCII text following these first two bytes. When writing a program that deals with BASIC's internal form, you need to consider the format of strings to avoid problems or bugs.

The original "Renumber" by Manny Juan rennumbers BASIC statements in RAM, resolves most line number references, and stays in memory for reuse.

"Renumber Plus" is a BASIC utility that enhances Manny Juan's Renumber. Renumber Plus does the following four operations the original Renumber does not:

- Resolves literal line number references after the LIST command.
- By-passes strings embedded in a statement.
- Resolves literal references following symbolic ones in a list of references.
- Allows you to choose where renumbering begins. These features add much to an already effective and useful tool.

Using Renumber Plus

1. Type Renumber Plus into your Atari.
2. Save the program with the direct command LIST "C" or LIST "D:REN". Using the LIST command allows you to merge Renumber Plus with your program.
3. LOAD your program into the Atari. The highest line number must be less than 32100. The last statement must be END, STOP or RETURN. LOADING your program erases Renumber Plus from memory.

4. Enter Renumber Plus into the Atari with the direct command ENTER "C:" or ENTER "D:REN".
5. Type GOTO 32100.
6. When the prompt BEGIN, FROM, BY appears, enter the following:
 - a. Beginning line number,
 - b. New starting line number, and
 - c. Increment value.
7. Enjoy the musical interlude while your Atari works. Do not press BREAK or RESET while the program rennumbers. The new line number followed by SR is displayed for each symbolic reference in your program. The new line number followed by NR is displayed for each reference to an old line number that does not exist.
8. When Renumber Plus finishes renumbering, the number of renumbered lines and the following message are displayed:
LIST "C:,"bbbb,eeee
bbbb = the first new line number
eeee = the last new line number
9. In order to save a copy of your renumbered program without the Renumber Plus program appended to it, use the LIST command (LIST "C:,"bbbb,eeee for cassette and LIST "D:
filename,"bbbb,eeee for disk).

Renumber Plus

```

32100 REM RENUMBER PLUS
32110 T8=256:I=1:Z=32100
32120 WM=0:X=PEEK(138)+PEEK(139)*T8:Y=PEEK(1
34)+PEEK(135)*T8+8*(PEEK(X+5)-128)+2
32130 ? "BEGIN, FROM, BY": INPUT ST, FR, BY: ? CHR
$(125)
32140 B=PEEK(136)+PEEK(137)*T8:X=B:M=FR
32142 LN=PEEK(X)+PEEK(X+I)*T8
32144 IF ST>FR AND LN-ST THEN ST=LN
32150 LN=PEEK(X)+PEEK(X+I)*T8:SOUND 0, LN, 10,
8
32160 IF LN=Z THEN 32220
32170 PL=PEEK(X+2):C=X+3
32180 LL=PEEK(C):C=C+I
32190 GOSUB 32280
32200 IF LL<PL THEN C=X+LL:GOTO 32180
32210 X=X+PL:M=M+BY*(LN>=ST):GOTO 32150
32220 M=FR:X=B:SOUND 1,0,0,0
32230 LN=PEEK(X)+PEEK(X+I)*T8:SOUND 0,-LN+32
768,10,8

```

```
32240 IF LN=Z THEN 32550
32245 IF LN<ST THEN 32270
32250 MH=INT(M/T8):ML=M-MH*T8
32260 POKE X,ML:POKE X+I,MH
32270 M=M+BY*(LN>=ST):X=X+PEEK(X+2):GOTO 322
30
32280 TK=PEEK(C)
32290 IF TK=10 OR TK=11 OR TK=12 OR TK=13 OR
TK=35 THEN C=C+I:GOSUB 32450:RETURN
32300 IF TK<>30 THEN 32345
32310 C=C+I:D=PEEK(C)
32320 IF D=23 OR D=24 THEN 32350
32330 IF D=14 THEN C=C+6
32335 IF D=15 THEN C=C+PEEK(C+I)+I
32340 GOTO 32310
32345 IF TK<>4 THEN 32380
32350 C=C+I:GOSUB 32450
32355 D=PEEK(C)
32360 IF D=18 THEN 32350
32362 IF D=14 THEN C=C+6
32364 IF D=15 THEN C=C+PEEK(C+I)+I
32366 IF D<>20 AND D<>22 THEN C=C+I:GO TO 32
355
32370 RETURN
32380 IF TK<>7 THEN RETURN
32390 C=C+I:D=PEEK(C)
32400 IF D=27 THEN 32430
32410 IF D=14 THEN C=C+6
32415 IF D=15 THEN C=C+PEEK(C+I)+I
32420 GOTO 32390
32430 C=C+I:IF C<(X+LL) THEN GOSUB 32450
32440 RETURN
32450 D=PEEK(C):IF D=20 OR D=22 THEN C=C+I:R
ETURN
32460 IF D<>14 THEN ? M;" SR,";:C=C+I:RETURN
32465 DD=PEEK(C+7):IF DD<>18 AND DD<>20 AND
DD<>22 THEN ? M;" SR,";:C=C+I:RETURN
32470 C=C+I:FOR J=0 TO 3:POKE Y+J,PEEK(C+J):
NEXT J
32480 IF WM<LN THEN WX=B:RN=FR:GOTO 32500
32490 WX=X:RN=M
32500 WN=PEEK(WX)+PEEK(WX+I)*T8:SOUND 1,WN,1
0,8
32510 IF WN<Z AND WN<WM THEN RN=RN+BY*(WN>=S
T):WX=WX+PEEK(WX+2):GOTO 32500
32520 IF WN<>WM THEN ? M;" NF,";:GO TO 32540
32525 IF WN<ST THEN 32540
```

```
32530 WM=RN:FOR J=0 TO 3:POKE C+J,PEEK(Y+J):  
      NEXT J  
32540 C=C+6:RETURN  
32550 ? :? (M-FR)/BY;" LINES"  
32560 ? "LIST";CHR$(34);"C:";CHR$(34);",";FR  
      ;",";M-BY  
32570 END
```

Purge

Al Casper

For the Atari 800 with 810 disk drive, this is a quicker and simpler method of housecleaning diskettes.

One of my favorite chores used to be clearing files off my diskettes, making room for new programs and files. Of course I'm kidding; I dreaded purging diskettes. First you had to load DOS and wait. Filenames had to be carefully entered, and finally the DELETE D:SLOW ? Y or N had to be dealt with. You also had to add one more step if the file was locked, or do it over from the start if you made a mistake. Repeat the above steps for each file you want deleted, and the entire process can easily take 20 minutes per diskette. "Purge" was written to make this job fast and easy, freeing your valuable time for other things.

Free Directory

When Purge is finished clearing your diskette, a directory is printed on the screen. The directory has two advantages over the DOS directory. First, you do not need to load DOS to use it. Second, the files are printed in two columns, allowing twice as many files to be displayed before they start scrolling off the top of the screen.

The program is written in two short sections, which makes it easy to save the DIR (Section A) as a separate program. The REMarks at the end of section A will explain this in more detail. I keep a copy of DIR on each of my diskettes. It requires only three sectors of disk space, well worth the time it can save you. I also have a LISTed version of DIR on a file named EDIR. I simply ENTER "D:EDIR" with any program I happen to have in memory. The high line numbers will almost never cause a conflict. Just type GOTO 32100 for a directory listing. DIR will now be a part of the program.

To use Purge, simply load the program, insert the diskette to be purged into disk drive one, and type RUN. One at a time the files on that diskette will be displayed on the screen. Pressing RETURN will display the next file. When an unwanted file is displayed, press CONTROL <P> to purge it. This process continues

until all the files have been displayed. Don't panic if you make an error along the way; just press BREAK and start over. The purging takes place after all the files have been displayed and only if you press P, as prompted on the screen. You'll hear a lot of action from the disk drive as the purging is taking place. The length of this operation varies with the number and length of files being deleted.

XIO Examples

The following is a line by line description of my program. This will be of most interest to programmers with limited experience working with disk operations. The XIO feature is the key to Purge. Writing this program in BASIC would have been very difficult without XIO. Note that the program listing does not have all the lines in correct order.

Line 32100 This special OPEN will allow inputs from the disk directory. The "*" in the filename is the same as a wildcard in DOS.

Line 32102 The TRAP is very useful. In this case it will detect the EOF (end of file), treat it as an error, and end the inputs.

Line 32104-32106 These are the INPUT(s) from the directory. The directory is printed in two columns.

Line 32110-32115 The file is CLOSED, and the program goes into an endless loop to prevent possible information from scrolling off the screen.

Line 32000 Another TRAP for EOF. The keyboard (K:) is OPENed for input.

Line 32004 The OPEN is again to the directory.

Line 32006 One at a time each directory entry is INPUT and tested for FREE SECTORS, which would be the last entry. The entry is then printed on the screen.

Line 32008 The program waits for an input from the keyboard. A chime sounds and slows things a bit.

Line 32010 If a purge was requested, the filename is created from the directory information.

Line 32012 The filename is saved in a larger string for later purging.

Line 32016-32017 Blank spaces have to be removed from the filename before they can be unlocked and deleted.

Line 32018 The XIO's perform unlock and delete just as if you were using DOS.

Line 32020 Files are CLOSED, and the DIR routine will follow.

Program 1. Section A: DIR

```

32050 REM SECTION (A) DIR
32055 REM
32060 REM WHEN FINISHED TYPING THIS SECTION
    SAVE IT WITH THE FILE NAME 'D:DIR'.
32065 REM ALSO LIST IT TO THE DISKETTE WITH
    THE FILE NAME{3 SPACES}'D:EDIR'.
32067 REM 'EDIR' CAN THEN BE ENTERED AT ANY
    TIME TO ATTACH A 'DIR'
32068 REM TO YOUR PROGRAM TO BE CALLED WITH
    A 'GOTO 32100'.
32070 REM THEN CONTINUE ADDING SECTION (B)
    TO SECTION (A)
32100 OPEN #5,6,0,"D:*.*)"
32102 CLR :GRAPHICS 0:POKE 82,1:DIM ENT$(17)
    :TRAP 32110: ? " DISK DIRECTORY
    {21 SPACES}"
32104 INPUT #5,ENT$: ? ENT$;"{4 SPACES}";
32106 INPUT #5,ENT$: ? ENT$:GOTO 32104
32110 CLOSE #5: ? " {4 SPACES} END
    {8 SPACES} PRESS BREAK{5 SPACES} END
    {3 SPACES}";:POKE 82,2
32115 GOTO 32115

```

Program 2. Section B: Purge

```

31900 REM SECTION (B) PURGE
31910 REM
32000 TRAP 32013:OPEN #4,4,0,"K:":DIM E$(17)
    ,S$(500),PG$(14):X=1:Y=14
32002 GRAPHICS 0: ? "{DOWN}TO PURGE": ? "
    {DOWN}AFTER EACH FILE DISPLAYED PRESS"
    : ? "CONTROL-P TO DELETE OR{3 SPACES}PR
    ESS RETURN"
32004 ? "TO CONTINUE":OPEN #5,6,0,"D:*.*)"
32006 INPUT #5,E$:POSITION 2,10:IF E$(5,16)<
    >"FREE SECTORS" THEN ? E$: ? " CHOICE
    [":GOTO 32008
32007 GOTO 32013
32008 GET #4,K:IF K<>16 THEN POSITION 2,12: ?
    " CHOICE ":FOR Q=15 TO 0 STEP -0.2:S
   OUND 0,20,10,Q:NEXT Q:GOTO 32006
32010 PG$(1,2)="D:":PG$(3,10)=E$(3,10):PG$(1
    1,11)=" ":PG$(12,14)=E$(11,13)
32012 S$(X,Y)=PG$:X=X+14:Y=Y+14:FOR Q=15 TO
    0 STEP -0.2:SOUND 0,40,10,Q:NEXT Q:GOT
    O 32006

```



```
32013 POSITION 2,15: ? "PRESS P TO PURGE";:FOR  
R Q=1 TO 120:POKE 764,255:NEXT Q:GET #  
4,K:IF K=80 THEN 32015  
32014 GOTO 32020  
32015 X=1:Y=14:S=0  
32016 TRAP 32020:PG$=S$(X,Y):FOR Q=1 TO 13:S  
=S+1:IF PG$(S,S)=" " THEN PG$(S,14)=PG  
$(S+1,14):S=S-1  
32017 NEXT Q  
32018 XIO 36,#3,0,0,PG$:XIO 33,#3,0,0,PG$:X=  
X+14:Y=Y+14:S=0:GOTO 32016  
32020 CLOSE #4:CLOSE #5
```

6 Advanced Techniques

6

Starshot

■ Matt Giwer

As this game will demonstrate, Atari BASIC can be fast enough if you know how to speed it up. Requires 24K and game paddles.

Atari graphics approach those available in dedicated graphics-oriented computers. Atari BASIC allows very fast manipulation of strings, Direct Memory Access for the Player/Missile Graphics, and the direct call of machine language from BASIC. This game combines all of these features and a few others.

Let's start the discussion of this program with the subroutine at line 30000. The first thing to do is to enable the Player/Missile Graphics.

Appendix A of the *Atari Hardware Manual* gives a detailed example of how to do this. This method works only when there is nothing on the screen; as soon as you write to the screen, the method fails. The usual approach is to reserve enough pages for the screen RAM, the Player/Missile graphics pages, etc. All in all, to use Player/Missile Graphics with GRAPHICS 7, you wind up reserving 32 pages and, in the process, taking care of the computer rather than letting the operating system (OS) take care of you. Here is how to do it right.

RAMTOP

Contained in register 106 is the number of pages of RAM available to you for your use after everything needed for the system has been accounted for. What we want to do is to change this number so that RAM is protected for the Player/Missile Graphics pages. This is accomplished by POKE 106, PEEK(106)-16. This puts a number into that register that is 16 pages less than the number the operating system determines upon powering up the computer or upon system reset. But just POKEing a new number does nothing until the computer makes use of it.

The second GRAPHICS 7 call causes the operating system to make use of this new RAMTOP to relocate the screen RAM and the display list below RAMTOP. If you do not make this graphics call, you will find that the screen memory is above the new, lower protected memory limit, and the system will crash at the first

attempt to scroll the screen. In other words, your system registers that point to the first screen byte, and the display list will be above RAMTOP. The operating system cannot handle this.

You proceed as normal but much more cleanly now that you have lowered the effective top of your RAM and made the operating system reorganize itself around that new maximum RAM with the second graphics call. Lines 30204 and 30206 are the enabling POKES for Player/Missile Graphics as described in many articles and in *De Re Atari*. Line 30208 is the POKE to tell the operating system where to find the start of the Player/Missile data. The start of this data is now simply RAMTOP.

With Player/Missile Graphics set up this way, you can forget about what the rest of the system is doing and treat it just as though Player/Missile Graphics were not in use. The operating system will take care of you.

Player Definition

The next routine of interest is at line 30236. (This is the machine language routine published in *COMPUTE!'s First Book of Atari Graphics*, page 164.) It provides relocation of the four players at machine language speeds by means of two POKES and, since the routine is executed during the vertical blanking time, the motion appears to be continuous. The rest of the 30000 lines define the players. Note that the RESTORE in line 30310 makes Player 3 the same as Player 2, although it is defined as a different color in line 30230.

Now let's jump to lines 100-120—we will get to the earlier lines later. These lines are the definitions that will be used later for named subroutines. The use of named subroutines is a desirable feature that greatly aids program development.

Lines 1890-1930 are both the one-time calls and those such as DISPLAY that are needed to set up the game at the start.

The subroutine at line 10000 draws the background in the way that makes this illusion of motion possible. Note that each set of lines is drawn with a different COLOR and that the COLOR numbers rotate 1, 2, 3, 1, 2, 3, and so forth. I will get back to this in a minute.

Color Rotation Simulates Motion

The START subroutine at line 5000 POKES numbers into the color registers so that you can see the screen and draws the eight attackers. You will also note that COLOR J also rotates the

COLOR assigned to the attacker graphic although in a more complex manner than in BACKGROUND.

The DISPLAY subroutine at line 6300 controls the scoring and number of lives information that will be shown in the bottom alphanumeric window.

ASELECT at line 6500 picks the order in which the attackers will attack from among the predefined ATTACK1-4\$ in lines 54 and 60.

Within the infinite loop at line 2100 you'll find the reason why I used different COLORS to draw the background. The four statements in line 2110 rotate the colors used in the background through the registers in a bucket brigade manner; the colors seem to be moving toward you. Given the drawn background, it appears that you are moving forward through the trench. This illusion of motion requires the use of three different colors as a minimum. If there were only two colors, they would appear to flicker back and forth rather than move. The instructions in this line will be used in almost every subroutine so that this illusion of motion is maintained.

This technique is useful in many applications—you can simulate many kinds of motion. If you were to reverse the order of the instructions, you would have the illusion of going backwards. Line 2120 is simply a short delay.

Another line that you will find throughout the program is first used at line 5017. $A = 74 + \text{PADDLE}(0)/2.92$ is the equation that limits the motion of Player 0 on the screen. The farthest left X location that Player 0 can move to is 74. The range of values for the PADDLE(0) is 0 to 228. Dividing this range of values by 2.92 converts the largest value of 228 to the rightmost location of Player 0 and makes the full left-to-right motion of the Player a full turn of the PADDLE. In order to simulate continuous motion, this equation is also put into every subroutine where the program execution takes a noticeable amount of time.

The subroutine MOVE at line 5100 is a loitering loop that waits a random number of loops until the first attack begins. When the number 50 is reached, program execution jumps to SELECT at line 5200.

The SELECT subroutine picks the sequence of the attackers from ATTACK1\$ through ATTACK4\$. ATTACK\$ for the first wave was initially called in line 1930. This routine randomly picks one of the four attack sequences defined in lines 54 and 60. An attempt

to read the ninth element in this string is TRAPped to line 5211, which redraws the attackers and starts over.

Note this use of the TRAP instruction. It is not meant simply to avoid a program crash, but rather to perform an integral program function. Rather than a RAM and time-consuming test or loop, one simple statement is used.

Lines 5215-5240 erase the chosen attacker, position Player 1 over the erased attacker, and give some warning sounds. Line 5241 calls the subroutine JOIN at line 5800. This routine adds together the strings which are used to define the X and Y positions of Player 1 as it moves from its initial position to its attack position.

Special TRAPs

The strings are the AX1\$ and AY1\$ through AX8\$ and AY8\$ that were defined back in the beginning of the program. These are the X and Y coordinates to be POKEd into PLX + 1 and PLY + 1. They are stored as groups of three numbers. These values are read in lines 5260-5270. Note that by using TRAP here I do not have to keep track of the number of elements in the string. And again instead of some test or loop, a simple statement is used. These strings are merely added together. No matter what the sequence of the attack, the last pattern is always the same, and the last set of numbers in the string is always the same.

The ATTACK subroutine at line 5300 is where the shooting occurs. The first call is for the subroutine PATTERN at line 5600. This subroutine chooses among five possible X position patterns and five possible Y position patterns. These are the rest of the strings defined in the beginning of the program. This independent choice of X and Y patterns permits a total of 25 different attack patterns.

In line 5315, the X and Y values for this attack motion are read out in groups of three. In this case, the TRAP is used to jump back to the PATTERN subroutine call to pick another pair of strings when the end of the STRING is reached. This gives continuously varying motion to the attacker.

Lines 5324 and 5325 change the size of the attacker as it comes closer or goes farther away. F and G are flags that control the firing and motion of the missiles. It is worth examining how these flags function.

F controls the attacker's missile firing. Other than its house-keeping function, the primary purpose of the IF F = 0 is to fix the

X and Y location at the moment of firing so that the motion is calculated only from this point. After F is set to 1, these statements are no longer executed. If they were, the missile would weave back and forth in X and Y in unison with the attacker. Behind the F = 1 flag are the calculations that determine whether the missile passes to the left or to the right. The G flag performs a similar program function.

Lines 5350 and 5352 check for missile-to-player collisions and direct action to the appropriate subroutine. Line 5355 clears the collision registers.

HITYOU, HITME, HITUS

The HITYOU, HITME, and HITUS subroutines introduce Players 2 and 3 as the explosions. In HITYOU and HITME, these two players are sequentially put in the same location as the hit player. This sequence is controlled by the TT variable. Note that the two explosion shapes are the same but of different colors. Also, when they are called, they are placed one Y position different. The purpose is to give some illusion of a dynamic explosion.

Lines 5440 and 5540 move the hit player and explosions off the screen. The logical truth statements determine whether the hit player was to the left or right of center when hit and then move it off the screen to the left or right as appropriate. Lines 5545 and 5547 cause the attacker and the explosions to grow larger as they go by.

The significant difference in the two subroutines is that in HITYOU there is an additional collision test in line 5560. This requires you to get out of the way of the hit player as it rolls off the screen. If you don't, you are also destroyed, and both players roll off the screen. This is controlled by the HITUS subroutine. Being hit by the attacker's missile and by the damaged attacker causes you to lose one life.

Good Practice

This is a quick review of a fairly complex program. It exploits many of the Atari's features. The method of reserving the Player/Missile Graphics pages by moving RAMTOP lets the machine take care of you and perhaps completes the official Atari version of how to turn on the function.

Starshot

```

40 J=66:PX=5
50 DIM ATTACK$(8),AX5$(J),AY5$(J),AX$(3*J),A
  Y$(3*J),APX1$(J),APY1$(J),APX$(J),APY$(J)
51 DIM AX4$(J),AY4$(J),APX2$(J),APY2$(J),APX
  3$(J),APY3$(J),APX4$(J),APY4$(J),APX5$(J)
  ,APY5$(J)
52 DIM AX3$(J),AY3$(J),AX2$(J),AY2$(J),AX6$(
  J),AY6$(J),AX7$(J),AY7$(J),AY8$(J),AX8$(J)
  ,AX1$(J),AY1$(J)
53 DIM PLAYER$(10),ATTACK1$(8),ATTACK2$(8),A
  TTACK3$(8),ATTACK4$(8)
54 ATTACK2$="37628415":ATTACK3$="28647135":A
  TTACK4$="47618325"
60 ATTACK1$="54637281":PLAYER$="1 2 3 4 5"
61 AX5$="13613613513413313213113012912812712
  6124122121121122123124125126126"
62 AY5$="03803703503403403403503703904104304
  5047049052056059062065068071074"
63 AX4$="11812012212412612813013213413413213
  0128126126126126126126126126126"
64 AY4$="03603403203002803003203403704004305
  0057063070076082080078076075074"
65 AX6$="156154152150148146144142140138136"
66 AY6$="038036034033034036038040042040038"
67 AX2$="078080082084086088090092094096098"
68 AY2$="038042044046048050052049046042038"
69 AX1$="058060062064066068070072074076078"
70 AY1$="038035031035038042046048046042038"
71 AX3$="098100102104106108110112114116118"
72 AY3$="040044048046044042040038036037038"
73 AX7$="176174172170168166164162160158156"
74 AY7$="038036034032030033036039042040038"
75 AX8$="196194192190188186184182180178176"
76 AY8$="040044048046044042040038036036038"
83 APX1$="1261201141101101141201261321381421
  42138132126120114110110114120126"
84 APY1$="0740770820900951001041051071091121
  14112109107105104100095090082077"
85 APX2$="1261281301341381421421361301241211
  18110107104107110118120124126128"
86 APY2$="0740790840860880941001061101141101
  08106100094087080080080078076075"
87 APX3$="1261301341381421461421381341301261
  26130134138142144142138134130126"
88 APY3$="0740740740740740820860900981061141
  20114106098090086082074074074074"
89 APX4$="1261341421341261181101101261341421
  34126118110110126134142132126126"

```

```

90 APY4$="0740780820860920860820780740780820
86092096092088084080076072072074"
91 APX5$="1261321381441501561621561501441381
321261201161110104098104110116126"
92 APY5$="0740700680700740800840900961021061
02096092086082078076074070072074"
100 BACKGROUND=10000:START=5000:MOVE=5100:SE
LECT=5200:ATTACK=5300:HITME=5400:HITYOU=
5500
110 PATTERN=5600:RESET=5700:JOIN=5800:HITUS=
5900
120 XSCR=6000:YSCR=6100:LOSS=6200:DISPLAY=63
00:RESET2=6400:ASELECT=6500
1890 GOSUB 30000
1900 GOSUB BACKGROUND
1910 GOSUB START
1920 GOSUB DISPLAY
1930 GOSUB ASELECT
2000 REM CONTROL LOOP
2100 FOR IJK=1 TO 2 STEP 0
2110 TEMP=PEEK(710):POKE 710,PEEK(709):POKE
709,PEEK(708):POKE 708,TEMP
2120 Q=SIN(1)
2130 GOSUB MOVE
2900 NEXT IJK
5000 REM START
5005 POKE 708,10:POKE 709,0:POKE 710,56:POKE
PLY,150:POKE 53761,132:REM 709,152
5010 FOR I=1 TO 8
5011 FOR J=0 TO 2
5016 TEMP=PEEK(710):POKE 710,PEEK(709):POKE
709,PEEK(708):POKE 708,TEMP
5017 A=74+PADDLE(0)/2.92:POKE PLX,A:POKE 537
60,A-33
5019 COLOR J*I:IF J*I=4 OR J*I=0 OR J*I=8 OR
J*I=12 OR J*I=16 THEN COLOR 1
5020 PLOT 20*I-10,J:DRAWTO 20*I-11,J
5021 COLOR J*I:IF J*I=4 OR J*I=0 OR J*I=8 OR
J*I=12 OR J*I=16 THEN COLOR 2
5022 PLOT 20*I-8,J+3:DRAWTO 20*I-12,J+3
5025 TEMP=PEEK(710):POKE 710,PEEK(709):POKE
709,PEEK(708):POKE 708,TEMP
5033 COLOR J*I:IF J*I=4 OR J*I=0 OR J*I=8 OR
J*I=12 OR J*I=16 THEN COLOR 3
5034 PLOT 20*I-8,J+6:DRAWTO 20*I-9,J+6:PLOT
20*I-12,J+6:DRAWTO 20*I-11,J+6
5036 NEXT J:NEXT I
5090 RETURN
5100 REM MOVE

```

```

5105 FOR IJK=1 TO 2 STEP 0
5110 TEMP=PEEK(710):POKE 710,PEEK(709):POKE
709,PEEK(708):POKE 708,TEMP
5111 A=SIN(1)
5120 A=74+PADDLE(0)/2.92:POKE PLX,A:POKE 537
60,A-33
5130 RR=RR+1:IF RR=50 THEN GOSUB SELECT:RR=I
NT(40*RND(0)):POKE 53763,0:POKE 53761,1
32
5185 NEXT IJK
5190 RETURN
5200 REM SELECT
5205 JJJ=JJJ+1
5210 TRAP 5211:R=VAL(ATTACK$(JJJ,JJJ)):COLOR
0:GOTO 5215:TRAP 40000
5211 GOSUB START:JJJ=0:GOTO 5205
5215 FOR J=0 TO 2
5220 PLOT 20*R-10,J:DRAWTO 20*R-11,J
5223 TEMP=PEEK(710):POKE 710,PEEK(709):POKE
709,PEEK(708):POKE 708,TEMP
5224 A=74+PADDLE(0)/2.92:POKE PLX,A:POKE 537
60,A-33
5225 PLOT 20*R-8,8-J:DRAWTO 20*R-9,8-J:PLOT
20*R-12,8-J:DRAWTO 20*R-11,8-J
5230 NEXT J
5235 PLOT 20*R-8,3:DRAWTO 20*R-12,3:PLOT 20*
R-8,5:DRAWTO 20*R-12,5
5236 POKE PLX+1,36+20*R:POKE PLY+1,38:PLOT 2
0*R-8,4:DRAWTO 20*R-12,4
5238 FOR Z=250 TO 50 STEP -50:FOR X=15 TO 0
STEP -5:SOUND 3,Z,8,X:NEXT X
5239 TEMP=PEEK(710):POKE 710,PEEK(709):POKE
709,PEEK(708):POKE 708,TEMP
5240 NEXT Z
5241 GOSUB JOIN
5249 TEMP=PEEK(710):POKE 710,PEEK(709):POKE
709,PEEK(708):POKE 708,TEMP:POKE 53763,
134
5250 A=86+PADDLE(0)/2.92:POKE PLX,A:POKE 537
60,A-33
5255 FOR J=1 TO 200
5260 TRAP 5280:X=VAL(AX$(J*3-2,J*3)):Y=VAL(A
Y$(J*3-2,J*3)):POKE PLX+1,X:POKE PLY+1,
Y:TRAP 40000:POKE 53762,Y-20
5265 TEMP=PEEK(710):POKE 710,PEEK(709):POKE
709,PEEK(708):POKE 708,TEMP
5266 A=74+PADDLE(0)/2.92:POKE PLX,A:POKE 537
60,A-33
5270 NEXT J

```

```

5280 GOSUB ATTACK:GOSUB RESET
5290 RETURN
5300 REM ATTACK
5305 GOSUB PATTERN
5310 FOR J=1 TO 200
5315 TRAP 5305:X=VAL (APX$(J*3-2,J*3)):Y=VAL (
    APY$(J*3-2,J*3)):TRAP 40000
5321 TEMP=PEEK(710):POKE 710,PEEK(709):POKE
    709,PEEK(708):POKE 708,TEMP
5322 A=74+PADDLE(0)/2.92:POKE PLX,A:POKE 537
    60,A-33
5324 IF Y>94 THEN POKE 53257,1:POKE 53258,1
5325 IF Y<94 THEN POKE 53257,0:POKE 53258,0
5330 POKE PLX+1,X:POKE PLY+1,Y:POKE 53762,Y-
    20
5333 IF F=0 THEN M1P=MYPMBASE+777+Y:POKE 532
    53,X:POKE M1P,12:M1PO=M1P:T=MYPMBASE+90
    7+Y:XT=X
5335 IF F=0 THEN F=1:POKE 53765,207:POKE 537
    64,100
5337 IF F=1 THEN M1P=M1P+7:XT=(-1.5+XT)*(XT<
    128)+(1.5+XT)*(XT>128):POKE 53253,XT:PO
    KE M1P,12:POKE M1PO,0
5338 IF F=1 THEN M1PO=M1P:POKE 53765,160:IF
    M1P>T-50 THEN F=0:POKE M1PO,0
5339 TEMP=PEEK(710):POKE 710,PEEK(709):POKE
    709,PEEK(708):POKE 708,TEMP
5340 IF G=0 THEN IF PTRIG(0)=0 THEN M0P=MYPM
    BASE+768+150:PT=80+PADDLE(0)/2.29:POKE
    M0P,3:G=1:POKE 53252,PT
5342 IF G=1 THEN M0PO=M0P:T0=M0P-70:G=2:POKE
    53765,15:POKE 53764,50
5347 IF G=2 THEN M0P=M0P-7:PT=(3.5+PT)*(PT<1
    28)+(-3.5+PT)*(PT>128):POKE M0P,3:POKE
    M0PO,0
5349 IF G=2 THEN POKE 53252,PT:M0PO=M0P:POKE
    53765,160:IF M0P<T0 THEN G=0:POKE M0PO
    ,0
5350 IF PEEK(53256)=2 THEN GOSUB HITYOU
5352 IF PEEK(53257)=1 THEN GOSUB HITME:POKE
    M0PO,0:POKE M1PO,0
5355 POKE 53278,0
5375 NEXT J
5380 POKE PLX,PADDLE(0):POKE PLY,148
5395 RETURN
5400 REM HITME
5405 POKE 53761,15:POKE M0PO,0:POKE M1PO,0:R
    R=0
5410 FOR J=1 TO 200

```

```

5412 IF TT=0 THEN POKE 53258,3:POKE PLY+2,14
      4+RR:POKE PLX+2,A:POKE PLX,A:POKE PLY,1
      48+RR:TT=1
5413 IF TT=1 THEN POKE 53259,3:POKE PLY+3,14
      4+RR:POKE PLX+3,A:POKE PLX,A:POKE PLY,1
      48+RR:TT=0
5415 TRAP 5410:X=VAL(APX$(J*3-2,J*3)):Y=VAL(
      APY$(J*3-2,J*3)):TRAP 40000
5421 TEMP=PEEK(710):POKE 710,PEEK(709):POKE
      709,PEEK(708):POKE 708,TEMP
5424 IF Y>94 THEN POKE 53257,1:POKE 53258,1
5425 IF Y<94 THEN POKE 53257,0:POKE 53258,0
5427 POKE PLX+1,X:POKE PLY+1,Y:POKE 53762,Y+
      20
5430 IF TT=0 THEN POKE 53258,3:POKE PLY+2,14
      4+RR:POKE PLX+2,A:POKE PLX+3,0:TT=1
5431 IF TT=0 THEN POKE 53258,3:POKE PLY+2,14
      4+RR:POKE PLX+2,A:POKE PLX,A:POKE PLY,1
      48+RR:TT=1
5432 IF TT=1 THEN POKE 53259,3:POKE PLY+3,14
      4+RR:POKE PLX+3,A:POKE PLX,A:POKE PLY,1
      48+RR:TT=0
5435 TEMP=PEEK(710):POKE 710,PEEK(709):POKE
      709,PEEK(708):POKE 708,TEMP
5440 RR=(RR+7):A=(A+7)*(A>128)+(A-7)*(A<127)
      :IF A<0 THEN J=201
5441 POKE 53760,RR
5442 IF A<0 OR A>255 THEN J=201
5444 IF 144+RR>255 THEN J=201
5490 NEXT J:GOSUB YSCR
5495 POKE PLY+2,229:POKE PLY+3,229:POKE 5376
      1,0
5497 RETURN
5500 REM HITYOU
5505 POKE 53763,15:POKE M0P0,0:POKE M1P0,0:R
      R=0:POKE M0P,0:POKE M1P,0
5510 FOR J=1 TO 200
5531 IF TT=0 THEN POKE PLY+2,Y-10:POKE PLX+2
      ,X:POKE PLY+1,Y:POKE PLX+1,X:POKE PLX+3
      ,0:TT=1
5532 IF TT=1 THEN POKE PLY+3,Y-9:POKE PLX+3,
      X:POKE PLY+1,Y:POKE PLX+1,X:POKE PLX+2,
      0:TT=0
5534 A=74+PADDLE(0)/2.92:POKE PLX,A:POKE 537
      62,Y:POKE 53760,41+PADDLE(0)/2.92
5540 Y=Y+7:X=(X+3.5)*(X>128)+(X-3.5)*(X<128)
5545 IF Y>94 THEN POKE 53257,1:POKE 53258,1:
      POKE 53259,1
5547 IF Y>130 THEN POKE 53257,3:POKE 53258,3
      :POKE 53259,3

```

```

5550 TEMP=PEEK(710):POKE 710,PEEK(709):POKE
    709,PEEK(708):POKE 708,TEMP
5560 IF PEEK(53260)<>0 THEN GOSUB HITUS
5582 IF Y>255 THEN J=201
5584 IF X>255 OR X<0 THEN J=201
5590 NEXT J:GOSUB XSCR
5595 POKE PL2+2,0:POKE PLX+3,0:POKE 53763,0
5597 RETURN
5600 REM SELECT PATTERN
5610 R=INT(5*RND(0))+1
5621 IF R=1 THEN APX$=APX1$
5622 IF R=2 THEN APX$=APX2$
5623 IF R=3 THEN APX$=APX3$
5624 IF R=4 THEN APX$=APX4$
5625 IF R=5 THEN APX$=APX5$
5626 TEMP=PEEK(710):POKE 710,PEEK(709):POKE
    709,PEEK(708):POKE 708,TEMP
5630 R=INT(5*RND(0))+1
5641 IF R=1 THEN APY$=APY1$
5642 IF R=2 THEN APY$=APY2$
5643 IF R=3 THEN APY$=APY3$
5644 IF R=4 THEN APY$=APY4$
5645 IF R=5 THEN APY$=APY5$
5690 RETURN
5700 REM RESET
5710 F=0:G=0:POKE 53257,0:POKE PLX+1,0
5790 RETURN
5800 REM JOIN
5810 IF R=1 THEN AX$=AX1$:AX$(LEN(AX$)+1)=AX
    2$:AX$(LEN(AX$)+1)=AX3$:AX$(LEN(AX$)+1)
    =AX4$
5812 IF R=1 THEN AY$=AY1$:AY$(LEN(AY$)+1)=AY
    2$:AY$(LEN(AY$)+1)=AY3$:AY$(LEN(AY$)+1)
    =AY4$
5815 IF R=2 THEN AX$=AX2$:AX$(LEN(AX$)+1)=AX
    3$:AX$(LEN(AX$)+1)=AX4$
5817 IF R=2 THEN AY$=AY2$:AY$(LEN(AY$)+1)=AY
    3$:AY$(LEN(AY$)+1)=AY4$
5820 IF R=3 THEN AX$=AX3$:AX$(LEN(AX$)+1)=AX
    4$
5822 IF R=3 THEN AY$=AY3$:AY$(LEN(AY$)+1)=AY
    4$
5825 IF R=4 THEN AX$=AX4$:AY$=AY4$
5830 IF R=5 THEN AX$=AX5$:AY$=AY5$
5835 IF R=6 THEN AX$=AX6$:AX$(LEN(AX6$)+1)=A
    X5$
5837 IF R=6 THEN AY$=AY6$:AY$(LEN(AY6$)+1)=A
    Y5$
5840 IF R=7 THEN AX$=AX7$:AX$(LEN(AX$)+1)=AX
    6$:AX$(LEN(AX$)+1)=AX5$

```

```

5842 IF R=7 THEN AY$=AY7$:AY$(LEN(AY$)+1)=AY
6$:AY$(LEN(AY$)+1)=AY5$
5845 IF R=8 THEN AX$=AX8$:AX$(LEN(AX$)+1)=AX
7$:AX$(LEN(AX$)+1)=AX6$:AX$(LEN(AX$)+1)
=AX5$
5847 IF R=8 THEN AY$=AY8$:AY$(LEN(AY$)+1)=AY
7$:AY$(LEN(AY$)+1)=AY6$:AY$(LEN(AY$)+1)
=AY5$
5890 RETURN
5900 REM HITUS
5905 POKE 53763,15:POKE M0P0,0:POKE M1P0,0:R
R=0:POKE M0P,0:POKE M1P,0
5910 FOR J=1 TO 200
5931 POKE PLY+2,Y-10:POKE PLX+2,X:POKE PLY+1
,Y:POKE PLX+1,X
5932 POKE PLY+3,Y-10:POKE PLX+3,A:POKE PLY,Y
:POKE PLX,A
5940 Y=Y+7:X=(X+3.5)*(X>128)+(X-3.5)*(X<128)
:A=(A+3.5)*(A>112)+(A-3.5)*(A<112)
5950 TEMP=PEEK(710):POKE 710,PEEK(709):POKE
709,PEEK(708):POKE 708,TEMP
5982 IF Y>255 THEN J=201
5984 IF X>255 OR X<0 THEN J=201
5990 NEXT J:GOSUB YSCR
5995 POKE PL2+2,0:POKE PLX+3,0:POKE 53763,0
5997 RETURN
6000 REM XSCR
6010 SCORE=SCORE+10
6080 GOSUB DISPLAY
6090 RETURN
6100 REM YSCR
6120 PLAYER$(2*PX-1,2*PX-1)=" "
6125 PX=PX-1
6130 IF PX=0 THEN GOSUB LOSS
6180 GOSUB DISPLAY
6190 RETURN
6200 REM LOSS
6210 IF SCORE>HSCR THEN HSCR=SCORE
6220 GOSUB DISPLAY
6280 GOSUB RESET2
6290 RETURN
6300 REM DISPLAY
6305 POKE 53258,0:POKE 53259,0
6310 ? PLAYER$
6320 ? "SCORE: ";SCORE
6330 ? "HIGH SCORE: ";HSCR
6340 IF PX=0 THEN ? " PUSH TRIGGER FOR ANOTH
ER GAME";
6350 IF PX=0 THEN IF PTRIG(0)=1 THEN 6350:GO
SUB RESET2:GOSUB ASELECT

```

```
6360 ? PLAYER$
6362 ? "SCORE: ";SCORE
6364 ? "HIGH SCORE: ";HSCR
6390 RETURN
6400 REM RESET2
6410 SCORE=0:PLAYER$="1 2 3 4 5"
6430 PX=5
6490 RETURN
6500 REM ASELECT
6510 ZZ=INT(4*RND(0))+1
6520 IF ZZ=1 THEN ATTACK$=ATTACK1$
6522 IF ZZ=2 THEN ATTACK$=ATTACK2$
6524 IF ZZ=3 THEN ATTACK$=ATTACK3$
6526 IF ZZ=4 THEN ATTACK$=ATTACK4$
6590 RETURN
10000 REM BACKGROUND
10005 FOR I=0 TO 3:POKE 708+I,0:NEXT I
10007 COLOR 0,20:DRAWTO 70,20:DRAWTO
70,40:DRAWTO 90,40:DRAWTO 90,20:DRAWTO
159,20
10010 COLOR 1:FOR I=1 TO 2
10020 PLOT 0,20+I:DRAWTO 70-I,20+I:DRAWTO 70
-I,40+I:DRAWTO 90+I,40+I:DRAWTO 90+I,2
0+I:DRAWTO 159,20+I:NEXT I
10040 COLOR 2:FOR I=1 TO 2
10050 PLOT 0,22+I:DRAWTO 68-I,22+I:DRAWTO 68
-I,42+I:DRAWTO 92+I,42+I:DRAWTO 92+I,2
2+I:DRAWTO 159,22+I:NEXT I
10060 COLOR 3:FOR I=1 TO 3
10070 PLOT 0,24+I:DRAWTO 66-I,24+I:DRAWTO 66
-I,44+I:DRAWTO 94+I,44+I:DRAWTO 94+I,2
4+I:DRAWTO 159,24+I:NEXT I
10080 COLOR 1:FOR I=1 TO 3
10090 PLOT 0,27+I:DRAWTO 63-I,27+I:DRAWTO 63
-I,47+I:DRAWTO 97+I,47+I:DRAWTO 97+I,2
7+I:DRAWTO 159,27+I:NEXT I
10100 COLOR 2:FOR I=1 TO 5
10110 PLOT 0,30+I:DRAWTO 60-I,30+I:DRAWTO 60
-I,50+I:DRAWTO 100+I,50+I:DRAWTO 100+I
,30+I:DRAWTO 159,30+I:NEXT I
10120 COLOR 3:FOR I=1 TO 5
10130 PLOT 0,35+I:DRAWTO 55-I,35+I:DRAWTO 55
-I,55+I:DRAWTO 105+I,55+I:DRAWTO 105+I
,35+I:DRAWTO 159,35+I:NEXT I
10140 COLOR 1:FOR I=1 TO 7
10150 PLOT 0,40+I:DRAWTO 50-I,40+I:DRAWTO 50
-I,60+I:DRAWTO 110+I,60+I:DRAWTO 110+I
,40+I:DRAWTO 159,40+I:NEXT I
10160 COLOR 2:FOR I=1 TO 7
```



```

10170 PLOT 0,47+I:DRAWTO 43-I,47+I:DRAWTO 43
-I,67+I:DRAWTO 117+I,67+I:DRAWTO 117+I
,47+I:DRAWTO 159,47+I:NEXT I
10180 COLOR 3:FOR I=1 TO 9
10190 PLOT 0,54+I:DRAWTO 36-I,54+I:DRAWTO 36
-I,74+I:DRAWTO 124+I,74+I:DRAWTO 124+I
,54+I:DRAWTO 159,54+I:NEXT I
10200 COLOR 1:FOR I=1 TO 12
10210 PLOT 0,63+I:DRAWTO 27-I,63+I:DRAWTO 27
-I,83+I:DRAWTO 133+I,83+I:DRAWTO 133+I
,63+I:DRAWTO 159,63+I:NEXT I
10220 COLOR 2:FOR I=1 TO 20
10230 PLOT 0,75+I:DRAWTO 14,75+I:PLOT 159,75
+I:DRAWTO 145,75+I:NEXT I
10300 RETURN
30000 REM *****PM SETUP*****
30010 GRAPHICS 7:POKE 106,PEEK(106)-16:GRAPH
ICS 7:POKE 752,1:REM *****16 PAGE RESE
RVE*****
30020 ? : ? : ? "{9 SPACES}PREPARE FOR COMBAT"
30204 POKE 53277,3:REM *****GRCTL PLAY&MISS
*****
30206 POKE 559,62:REM *****DMACTL,1LINE,PLAY
,MIS,NORM FIELD*****
30208 POKE 54279,PEEK(106):REM *****PMBASE I
S NOW RAMTOP*****
30210 POKE 53256,3:POKE 53257,0:POKE 53258,0
:POKE 53259,0:REM *****PLAY SIZES*****
30212 POKE 623,33:REM *****PRIORITY PL OVER
PF*****
30214 MYPMBASE=256*PEEK(106):REM *****NEW PM
BASE*****
30230 POKE 704,134:POKE 705,24:POKE 706,46:P
OKE 707,54:POKE 1788,(PEEK(106)+4):REM
*****START OF PM DATA*****
30232 POKE 710,52:POKE 709,58:POKE 711,29:PO
KE 712,0
30236 REM *****VBLANK INTERRUPT ROUTINE*****
30238 FOR I=1536 TO 1706:READ A:POKE I,A:NEX
T I
30240 FOR I=1774 TO 1787:POKE I,0:NEXT I
30242 DATA 162,3,189,244,6,240,89,56,221,240
,6,240,83,141,254,6,106,141
30244 DATA 255,6,142,253,6,24,169,0,109,253,
6,24,109,252,6,133,204,133
30246 DATA 206,189,240,6,133,203,173,254,6,1
33,205,189,248,6,170,232,46,255
30248 DATA 6,144,16,168,177,203,145,205,169,
0,145,203,136,202,208,244,76,87

```

```

30250 DATA 6,160,0,177,203,145,205,169,0,145
      ,203,200,202,208,244,174,253,6
30252 DATA 173,254,6,157,240,6,189,236,6,240
      ,48,133,203,24,138,141,253,6
30254 DATA 109,235,6,133,204,24,173,253,6,10
      9,252,6,133,206,189,240,6,133
30256 DATA 205,189,248,6,170,160,0,177,203,1
      45,205,200,202,208,248,174,253,6
30258 DATA 169,0,157,236,6,202,48,3,76,2,6,7
      6,98,228,0,0,104,169
30260 DATA 7,162,6,160,0,32,92,228,96
30262 S=USR(1696)
30276 PLX=53248:PLY=1780:PLL=1784
30278 POKE PLL,9:POKE PLL+1,8:POKE PLL+2,26:
      POKE PLL+3,26
30282 FOR I=MYPMBASE+1024 TO MYPMBASE+1032:R
      EAD A:POKE I,A:NEXT I:REM *****DEFENDE
      R PLAYER 0*****
30283 DATA 24,24,60,60,126,255,126,36,36
30285 FOR I=0 TO 7:READ A:POKE MYPMBASE+1280
      +I,A:NEXT I:REM *****ATTACKER PLAYER 1
      *****
30287 DATA 204,204,204,252,252,48,48,48
30299 REM *****EXPLOSION PLAYER 2*****
30300 FOR I=MYPMBASE+1280+256 TO MYPMBASE+25
      6+1305:READ A:POKE I,A:NEXT I
30305 DATA 24,36,80,52,90,52,105,93,170,237,
      181,106,253,94,171,246,173,85,44,90,11
      6,44,52,44,24,8
30309 REM *****EXPLOSION PLAYER 3*****
30310 RESTORE 30305:FOR I=MYPMBASE+1280+512
      TO MYPMBASE+1305+512:READ A:POKE I,A:N
      EXT I
30590 RETURN

```

Laser Gunner II

Gary R. Lecompte

Version for the Atari by Charles Brannon with revisions by Thomas A. Marshall.

This revised version of "Laser Gunner" mixes machine language and BASIC to make a very exciting game. The enhancements include having two missiles on the screen simultaneously and smooth animation even as the missiles are fired.

In your corner of the universe, a zone of high-pressure radioactive plasma is contained by a platinum-iridium wall. Your ship, immersed in the red zone, is charged with a vital duty: defend the wall. The vengeful enemies of your civilization send wave after wave of attack ships in an effort to breach the wall. These semi-smart robot ships will concentrate their firepower on your weakest spot and mercilessly try to fire their way into the wall.

Your only defense is your powerful particle beam which you use to fend off the attacking drones. The enemy ships are wary of your power, so if you move too close to an attack point, you can spook the enemy ship into picking another target. Move to shoot at the new position, and it will just cruise back to another vulnerable spot. You must not let the enemy blast a hole in the wall since, like a balloon stuck with a pin, the radioactive plasma will explode, reducing your ship to an expanding shell of iridescent particles.

As the laser gunner, you try to react quickly to your enemy's shots. Follow the ship as well as you can, but do not stray too far from a weak spot. When you destroy one ship, another will appear at a random position, and will home in on a vulnerable spot in the wall.

A Novel Player/Missile Technique

For a game written in BASIC, "Laser Gunner" is reasonably fast and smooth. The smoothness of motion comes from player/missile graphics, but the speed comes from an unusual technique that lets you move player/missile graphics at machine language speed.

A special graphics technique is used here. Instead of storing the player/missile graphics at the top of memory, a large string is dimensioned to hold the player/missile data. When a string is dimensioned, a block of memory is reserved for it. The starting address of the string can be determined by using the ADR function. The problem is that player/missile graphics must start on an even 1K boundary (the address must be a multiple of 1024), or a 2K boundary (divisible by 2048) for single-resolution player/missile graphics. Strings are given the next available address when dimensioned, which would only be on an even kilobyte address by sheer coincidence.

So when the AddRess of the string is determined, we must find what offset to add to the address to reach the next boundary. It can be shown that in worst case conditions (i.e., the address is just one byte past a 1K or 2K boundary), we must allow for an offset of at least 1023 bytes for double-resolution, or 2048 bytes for single-resolution P/M graphics. So, although double-resolution P/M graphics require only 1024 bytes, we must dimension the holding string at least 2048 bytes. Then, a simple calculation (lines 150-160) will give us the starting address within the string of the P/M base address, PMBASE. This value is then used to "set up" P/M graphics as usual.

The advantage of using a string is twofold: one, we know that BASIC is covetously protecting the string from the "RAMTOP Dragon" (see *COMPUTE!'s Second Book of Atari Graphics*) and other nasties. Second, we can use BASIC's fast string manipulation commands to move segments of strings around, scroll a string, erase a string, copy one string to another, and more. Since the memory being moved in the string is the P/M memory, these manipulations directly modify the players and missiles. And since these string operations internally proceed at machine language speed, we get fast P/M animation using BASIC. Although the code is not as straightforward as dedicated P/M commands such as PMMOVE or PMGRAPHICS, it sure beats cryptic USR statements. As a matter of fact, since BASIC permits such flexibility with strings, it may be the best solution to using P/M graphics from BASIC.

Using Vertical Blank for Smoother Motion

The original version of Laser Gunner required all other motion to stop when missiles were fired. By using a vertical blank interrupt routine, continuous and smooth motion can be achieved.

The vertical blank (VB) is the time during which the television's electron beam is turned off while it returns from the lower-right corner of the screen to the top-left. Depending on the graphics mode and other interrupts, there are approximately 7980 *machine cycles* available during a single VB. (A machine cycle is the smallest measurement of time on your computer's internal clock.)

Bringing VB into the Picture

To utilize the VB, we first have to tell the operating system (OS) where to go. We do this by performing a Vertical Blank Interrupt (VBI) through the Set Vertical Blank Vector (SETVBV) routine. Before jumping to the SETVBV, we have to load the least significant byte (LSB) in the Y register and the most significant byte (MSB) in the X register of our VB machine language routine.

Into the accumulator we can place either a 6 or a 7. Six is for deferred mode; the OS does its housekeeping operations before it executes our code. Seven is for immediate mode; the OS executes our code first during the VB. Since we will be checking the collision registers, we will be loading a 6 into the accumulator. The BASIC program initializes the SETVBV through the USR statement on line 1460. To return control to the OS, we jump back through \$E45F.

The BASIC and the machine language (ML) programs interact through several PEEKs and POKEs. The ML program checks the STRIG(0), location \$0284, for the press of a button, and moves both missiles horizontally. Since the player/missile graphics are defined in strings, it is easier to have BASIC draw and erase the missiles by PEEKing the flags that the ML program sets.

In the enhanced version, both missiles appear on the screen at the same time. This requires the additional coding located at \$06D7. The missiles are defined as:

BIT	7	6	5	4	3	2	1	0
	M3	M2	M1	M0				

Since it is difficult for Atari BASIC to selectively turn bits off and on, we will use ML to change the bits. The AND instruction is used to set bits to zero (off). ANDing a bit with zero sets the bit to zero. The ORA instruction is used to set bits to one (on). By ORAing a bit with one, we set the bit to one. The flipping of the missile bits is done in the subroutines at lines 1300-1330.

Further Enhancements

The programming technique of performing graphics movement during the vertical blank enhances Laser Gunner almost to the level of difficulty of professional arcade games. Further program execution speed can be achieved by removing the REMs and moving the part of the program that does most of the action to the beginning. This shortens the memory that BASIC has to search to find line number references. An additional enhancement would be to add a sound routine during the VB each time the trigger is pressed.

Laser Gunner II

```

0 REM  LASER GUNNER.V2(17 SPACES)
1 REM  An enhancement of Laser Gunner
2 REM  in Issue 30 of COMPUTE!(9 SPACES)
3 REM  The program allow simultaneous
   {3 SPACES}
4 REM  motion of the missiles using the
5 REM  verticle blank period.(10 SPACES)
6 REM  Developed by Thomas A. Marshall
7 REM  Albuquerque, NM 87123(11 SPACES)
10 GOSUB 1400
20 RESTORE
100 DIM PM$(2048):GRAPHICS 2+16
110 DIM ALIEN$(11),PLAYER$(11),NULL$(11),EXP
   LOD$(12*9),TARGET(20)
120 FOR I=1 TO 11:NULL$(I)=CHR$(0):NEXT I
130 LEVEL=15:CNT=15:REM DECREASE LEVEL FOR A
   HARDER GAME
140 A=ADR(PM$):REM RAW ADDRESS
150 PMBASE=INT(A/1024)*1024:REM NEAREST 1 K
   BOUNDARY
160 IF PMBASE<A THEN PMBASE=PMBASE+1024:REM
   IF BELOW STRING, GO TO NEXT 1K BOUNDARY
170 S=PMBASE-A:REM START OF PMBASE IN STRING
   (OFFSET)
180 POKE 559,46:REM SET DOUBLE-LINE RES.
190 POKE 54279,PMBASE/256:REM TELL ANTIC WHE
   RE PMBASE IS
200 POKE 53277,3:REM TURN ON PLAYER/MISSILE
   DIRECT MEMORY ACCESS(DMA)
210 PM$=CHR$(0):PM$(2048)=CHR$(0):PM$(2)=PM$
   :REM CLEAR OUT ALL P/M MEMORY
220 POSITION 4,0: ? #6;"laser gunner"
230 ? #6:FOR I=1 TO 10: ? #6;"■":NEXT I:POSIT
   ION 0,0

```

```

240 REM STRING POS OF PLAYER 0-3, AND MISSIL
    ES IN STRING:
250 P0=S+512:P1=P0+128:P2=P1+128:P3=P2+128:M
    S=S+384
260 PM$(P2+32)=CHR$(255):PM$(P2+127)=CHR$(25
    5):PM$(P2+33,P2+127)=PM$(P2+32):REM CREA
    TE WALL
270 PM$(P3,P3+127)=PM$(P2,P2+127):REM CREATE
    "ZONE"
280 POKE 53250,92:REM POSITION PLAYER 2, THE
    WALL
290 POKE 53251,60:REM POSITION PLAYER 3, THE
    ZONE
300 POKE 53258,0:POKE 53259,3:REM REM MAXIMU
    M WIDTH
310 POKE 706,14:POKE 707,66:REM SET COLOR OF
    PLAYERS 2 AND 3
320 DATA 0,8,28,62,255,62,255,62,28,8,0
330 FOR I=1 TO 11:READ A:ALIEN$(I)=CHR$(A):N
    EXT I:REM PLACE INTO STRING, HENCE INTO
    P/M MEMORY
340 AY=32:REM ALIEN VERTICAL LOCATION
350 PM$(P1+AY,P1+AY+11)=ALIEN$:REM PLACE INT
    O STRING INTO P/M MEMORY
360 POKE 705,6*16+10:REM SET COLOR OF ALIEN
    TO PURPLE
370 POKE 53249,180:REM SET HORIZNONTAL POSIT
    IN
380 POKE 53257,1:REM SET ALIEN TO DOUBLE-WID
    TH
390 REM SET UP EXPLODE$, USE FOR EXPLOSION O
    F ALIEN
400 FOR I=1 TO 108:READ A:EXPLODE$(I)=CHR$(A
    ):NEXT I:REM EXPLODE DATA
410 DATA 8,28,62,255,54,255,62,28,8,8,28,62,
    235,54,235,62,28,8,8,28,54,227,34,227,54
    ,28,8
420 DATA 8,24,34,227,34,227,18,24,8,8,24,34,
    194,32,163,18,8,8
430 DATA 0,0,0,0,24,24,0,0,0,0,0,0,32,8,24,0
    ,4,0,0,0,36,0,16,0,36,0,0,128,10,128,0
    ,16,0,16,65
440 DATA 0,9,0,0,32,0,32,0,8,0,0,0,64,0,0,64
    ,0,4,0,0,0,0,0,0,0,128,0
450 RY=INT(78*RND(0)+32):MH=190+RY*2:REM ATT
    RACT MODE:
455 POSITION 9,5: ? #6;"PRESS":POSITION 9,6: ?
    #6;"START"

```

```

460 FOR I=32 TO 110:PM$(P1+I,P1+I+11)=ALIEN$
:IF I=RY THEN PM$(MS+RY+10,MS+RY+10)=CHR$(12)
470 IF I>RY THEN POKE 53253,MH-I*2
480 IF PEEK(53279)>6 THEN NEXT I
490 PM$(MS+RY+10,MS+RY+10)=CHR$(0)
500 FOR I=110 TO 32 STEP -1:PM$(P1+I,P1+I+11)
=ALIEN$:IF PEEK(53279)>6 THEN NEXT I
510 IF PEEK(53279)>=7 THEN 450
515 POSITION 9,5:?"#6;"{5 SPACES}":POSITION
9,6:?"#6;"{5 SPACES}"
520 IF PEEK(53279)=3 THEN FOR I=0 TO 4:POKE
53248+I,0:NEXT I:GRAPHICS 0:END
530 DATA 0,0,224,48,120,63,120,48,224,0,0
540 FOR I=1 TO 11:READ A:PLAYER$(I)=CHR$(A):
NEXT I
550 PY=60:REM SET PLAYER'S VERITCAL LOCATION
560 PM$(P0+PY,P0+PY+11)=PLAYER$
570 PM$(P1,P1)=CHR$(0):PM$(P1+127,P1+127)=CH
R$(0):PM$(P1+2,P1+127)=PM$(P1)
580 AY=INT(78*RND(0)+32):PM$(P1+AY,P1+AY+11)
=ALIEN$:REM RESET ALIEN
590 POKE 53256,1:REM PLAYER 0 DOUBLE-WIDTH
600 POKE 53248,64:REM HORIZONTAL POSITION OF
PLAYER 0
610 POKE 704,26:REM COLOR OF PLAYER 0
620 POKE 53260,1:REM MISSILE 0 DOUBLE-WIDTH
630 ST=STICK(0):IF ST<>15 THEN DIR=ST:F=2:SO
UND 0,100,0,8
635 IF PEEK(CMPFLG)=1 THEN PM$(TMS,TMS)=CHR$(
0):POKE CMPFLG,0:REM THE MISSILES HIT E
ACH OTHER
636 IF PEEK(COLFLG)=1 THEN POKE COLFLG,0:GOT
O 900:REM THE ALIEN MISSILE HIT THE WALL
OR ZONE
640 PY=PY-(DIR=14)*(PY>32)*F+(DIR=13)*(PY<11
0)*F:F=1:REM UPDATE PLAYER
650 PM$(P0+PY,P0+PY+11)=PLAYER$:SOUND 0,0,0,
0
660 IF PEEK(M0FLG)=1 THEN GOSUB 1310:REM ERA
SE THE PLAYER'S MISSILE
670 IF PEEK(TRIGFLG)=0 THEN GOSUB 1310:POKE
M0FLG,0:TMS=MS+PY+5:GOSUB 1300:POKE TRIG
FLG,1:REM THE TRIGGER WAS PRESSED
720 IF PEEK(HITFLG)<>0 THEN 790:REM NO COLLI
SION
725 REM THE PLAYER'S MISSILE HIT THE ALIEN
730 SCR=SCR+10:POSITION 11-LEN(STR$(SCR))/2,
5:?"#6;SCR

```



```

735 PM$(TMS,TMS)=CHR$(0):POKE M0FLG,1:POKE H
    ITFLG,1:POKE 53278,0
740 AY=AY+1:P=PEEK(705):REM PRESERVE COLOR O
    F ALIEN
750 FOR I=0 TO 11:Z=I*9:PM$(P1+AY,P1+AY+9)=E
    XPLODE$(Z+1,Z+9)
760 POKE 705,PEEK(53770):POKE 53279,0:SOUND
    0,I*2,0,15-I:FOR W=1 TO 2:NEXT W:NEXT I
770 POSITION 5,5:PRINT #6;"{10 SPACES}":REM E
    RASE SCORE
780 SOUND 0,0,0,0:POKE 705,P:GOTO 570
790 IF AY=PY THEN 870:REM TOO CLOSE FOR COMF
    ORT
800 IF TARGET=0 THEN GOSUB 950:TARGET=TARGET
    (INDEX):REM SELECT A TARGET
810 IF AY<>TARGET THEN 840
820 CNT=CNT-1:IF CNT THEN 630
830 CNT=LEVEL:GOTO 870
840 AY=AY+SGN(TARGET-AY):REM MOVE TOWARDS TA
    RGET
850 PM$(P1+AY,P1+AY+11)=ALIEN$
860 GOTO 630
870 IF ABS(AY-PY)<10 THEN GOSUB 970
875 IF PEEK(ALIEFLG)=0 THEN 630
880 POKE ALIEFLG,0:TM1S=MS+AY+5:GOSUB 1320:T
    TAY=AY:GOTO 630
900 P=ASC(PM$(P2+TTAY+5))*2-256:GOSUB 1330:P
    OKE 53278,0:REM CUT HOLE IN WALL
910 IF P<0 THEN 990:REM WALL DESTROYED
920 PM$(P2+TTAY+5,P2+TTAY+5)=CHR$(P)
930 GOTO 630
940 REM PICK A TARGET
950 INDEX=INDEX+1:TARGET(INDEX)=INT(78*RND(0
    )+32):RETURN
970 IF INDEX=1 THEN 950
980 TARGET=TARGET(INT(INDEX*RND(0)+1)):RETUR
    N
990 REM DESTRUCTION OF PLAYER
1000 FOR I=1 TO 100:Z1=TTAY+5+I:Z2=TTAY+5-I
1005 PM$(TMS,TMS)=CHR$(0):POKE M0FLG,1:POKE
    M0PFLG,72
1010 IF Z1<126 THEN PM$(P2+Z1,P2+Z1)=CHR$(0)
1020 IF Z2>30 THEN PM$(P2+Z2,P2+Z2)=CHR$(0)
1030 IF Z1<126 OR Z2>30 THEN NEXT I
1040 FOR I=30 TO 1 STEP -1:FOR J=0 TO 20 STE
    P 3:SOUND 0,J+1,10,8:POKE 707,PEEK(5377
    0):NEXT J:NEXT I
1050 SOUND 0,0,0,0:SOUND 1,0,0,0:POKE 707,14
    :FOR W=1 TO 50:NEXT W:POKE 707,0

```

```

1060 FOR I=0 TO 15 STEP 0.2:SOUND 0,I,8,I:PO
KE 704,16+I:NEXT I
1070 SOUND 0,0,0,0
1080 Z1=PY:Z2=PY:INCR=0
1090 Z1=Z1+INCR*(Z1<128):Z2=Z2-INCR*(Z2>=0):
POKE 704,PEEK(53770)
1100 PM$(P0+Z1,P0+Z1)=CHR$(255):PM$(P0+Z2,P0
+Z2)=CHR$(255):POKE 53279,0
1110 INCR=INCR+0.5:IF Z1<127 OR Z2>0 THEN 10
90
1120 FOR I=1 TO 100:POKE 704,PEEK(53770):NEX
T I
1130 FOR I=0 TO 7:POKE 53248+I,0:NEXT I:GRAP
HICS 18
1140 POSITION 4,0:PRINT #6;"laser gunner":PO
SITION 3,5:PRINT #6;"your score was:";
1150 POSITION 10-LEN(STR$(SCR))/2,7:PRINT #6
;SCR
1160 FOR I=15 TO 0 STEP -0.2:SOUND 0,10+10*R
ND(0),0,I:SOUND 1,100+10*RND(0),16,I
1170 SETCOLOR 4,3,14*RND(0):NEXT I
1280 RUN
1299 REM M0 SET
1300 Q=USR(ANORA,ASC(PM$(TMS,TMS)),3,2):PM$(
TMS,TMS)=CHR$(Q):RETURN
1309 REM M0 CLEAR
1310 Q=USR(ANORA,ASC(PM$(TMS,TMS)),12,1):PM$
(TMS,TMS)=CHR$(Q):RETURN
1319 REM M1 SET
1320 Q=USR(ANORA,ASC(PM$(TM1S,TM1S)),12,2):P
M$(TM1S,TM1S)=CHR$(Q):RETURN
1329 REM M1 CLEAR
1330 Q=USR(ANORA,ASC(PM$(TM1S,TM1S)),3,1):PM
$(TM1S,TM1S)=CHR$(Q):RETURN
1400 TRIGFLG=1546:HITFLG=1547:M0FLG=1548:TMS
=1:TM1S=1
1410 ALIEFLG=1550:COLFLG=1551
1420 ANORA=1753:CMPLG=1553
1430 IF PEEK(1753)=104 THEN RETURN
1440 GRAPHICS 18: ? #6;"INITIALIZING"
1450 RESTORE 1500:GOSUB 1500
1460 A=USR(1536):RETURN
1500 FOR I=1536 TO 1552:READ A:POKE I,A:NEXT
I
1509 REM INIT 1536 TO 1552
1510 DATA 104,169,6,170,160,22,32,92,228,96,
1,1,1,72,1,0,180
1520 FOR I=1558 TO 1709:READ A:POKE I,A:NEXT
I

```

```
1530 REM MISSILE MOVING ROUTINE
1540 DATA 173,132,2,201,0,240,2,208,12,205,1
      2,6,240,12,169,0,141,10,6,240
1550 DATA 58,205,12,6,240,53,238,13,6,238,13
      ,6,173,13,6,141,4,208,173,8
1560 DATA 208,41,2,208,9,173,13,6,201,190,14
      4,27,176,15,173,13,6,201,170,144
1570 DATA 18,169,0,141,30,208,141,11,6,169,1
      ,141,12,6,169,72,141,13,6,173
1580 DATA 14,6,201,0,208,63,173,9,208,41,1,2
      08,21,173,9,208,41,12,208,29
1590 DATA 206,16,6,206,16,6,173,16,6,141,5,2
      08,208,35,169,1,141,17,6,141
1600 DATA 12,6,169,72,141,13,6,208,5,169,1,1
      41,15,6,169,0,141,30,208,169
1610 DATA 1,141,14,6,169,180,141,16,6,76,95,
      228
1620 FOR I=1753 TO 1791:READ A:POKE I,A:NEXT
      I
1630 REM END-OR ROUTINES
1640 DATA 104,104,104,141,215,6,104,104,141,
      216,6,104,104,201,1,208,9,173,215,6
1650 DATA 45,216,6,76,249,6,173,215,6,13,216
      ,6,133,212,169,0,133,213,96
1660 RETURN
```

The Cruncher

Andrew Lieberman

Many longer programs could benefit from this memory-saving technique, which saved 7,000 bytes in the music DATA within the author's music program.

Programs are written every day using DATA statements. Often the numbers in these statements are for SOUND and PLOT commands and happen to be in the range of 0 to 255. Frequently, the program loads these numbers into a matrix. This method of DATA storage is inefficient; it wastes *lots* of memory.

There is, however, a way to solve this problem, and an easy way to change already existing programs to a more compact form. Using the "Cruncher," I knocked 7K — that's right, 7000 bytes — off a music program. It took about 40 minutes, and that includes debugging. Many programs can easily be done in half that time.

Each character on the Atari has an ATASCII value ranging from 0 to 255. Look in your *BASIC Reference Manual*, Appendix C. Take, for example, the letter A. Its corresponding number is 65. By using this code, we can convert each number (using one to three digits) to a single character using only one character. It would be a very tedious process if we took each number, looked it up on the chart, and then replaced the number in a program with a single character.

That's where the Cruncher comes in. It won't do all of the work, but it will do most of it. We can further save memory by condensing all of these single characters into one large string instead of a matrix. This is the big memory saver: each character in a matrix takes about seven bytes, but in a string takes only one. So, pull out a program with a lot of numbers and let's get to work. (Note: This is not a standard procedure. Your program may require modifications of the process of conversion. Read through the procedure and think about what you are doing; otherwise, you may find yourself hopelessly lost.)

First, type the following subroutine into your Atari, and LIST it to cassette or disk. This way you can load it on top of the program to be converted.

```

0 A=PEEK(136)+PEEK(137)*256: ? "WHAT
  LINE";: INPUT X: TRAP 32003: GOTO 320
00
32000 LI=PEEK(A)+PEEK(A+1)*256: IF LI
<>X THEN A=A+PEEK(A+2): GOTO 32
000
32001 A=A+1: IF PEEK(A)=90 THEN READ
D: POKE A,D
32002 GOTO 32001
32003 END

```

Second, load the program to be converted. Put in a DIM statement and DIMension a string, say A\$, to the number of numbers in the DATA statements. If your program READs the DATA and then puts it in a matrix, get rid of the READ statements. Otherwise, change a routine like this:

```
100 FOR I=1 TO 100:READ A,B:PLOT A,B
  :NEXT I
```

to this:

```
100 FOR I=1 TO 100:A=ASC(A$(I,I)):B=
ASC(A$(I+1,I+1)):PLOT A,B:NEXT I
```

or better yet:

```
100 FOR I=1 TO 100:PLOT ASC(A$(I,I))
,ASC(A$(I+1,I+1)):NEXT I
```

If your program handles the DATA in a different way, then it's up to you to figure out the rest of that part on your own.

Now we are almost ready to convert the DATA. Before we can put the characters into A\$, we must have an A\$. It is already DIMensioned, but we must add space for the characters in the program. Get an idea as to approximately how many numbers are to be converted, say 200. Then type something like this into your program:

```
50 A$(1,50)="ZZZZZZZZZZZZZZZZZZZZZZZZZ  
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ"  
52 A$(51,100)="ZZZZZZZZZZZZZZZZZZZZZZZ  
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ"  
54 A$(101,150)="ZZZZZZZZZZZZZZZZZZZZZZZ  
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ"  
56 A$(151,200)="ZZZZZZZZZZZZZZZZZZZZZZZ  
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ"  
58 A$(201,225)="ZZZZZZZZZZZZZZZZZZZZZZZ  
ZZ"
```

It doesn't hurt to put in some extras; you can always take them

out later. To easily duplicate a line, just type it, press RETURN, move the cursor back to the line number, change it, and press RETURN. (Note: You *must* use capital Z's.) Once you have done this, type RUN. Tell the computer what line your Z's start at (in our sample, 50). Now, wait while the computer figures everything out. When READY appears, LIST the program and see what happens. *Voilà!* The Z's now look like a lot of garbage!

Fourth, and last, get rid of any extra Z's and delete line 0, lines 32000 to 32003, and all of the numerical DATA statements. Now type RUN and watch your program run faster than ever. Sit back and say to yourself, "Gee, that was easy. What program should I fix next?"

The Mystery Revealed

For those of you who would like to know how this program works, I will explain it step by step. The first thing the computer does is find out where the program is stored in RAM. By PEEKing addresses 136 and 137, the Cruncher finds out the first address of the program. The TRAP is so that when the computer is out of DATA, it ENDS without an error.

Next, the computer finds line X. The first three bytes of each line give very important information. The first two tell the line number, and the third tells the length. To check if we are at line X, we first find out at which line we are. If LI isn't equal to X, we must advance the pointer to the next line. We do this by adding the length of the line to our original number and trying again.

Now the conversion process begins. A loop begins that checks each address to see if it is 90, or a Z. If it is, the program READs a piece of DATA and POKEs it into the program. We then loop back and continue the process. When we run out of DATA, the TRAP is sounded and the program ENDS.

PEEK and POKE Alternatives

Jerry White

This tutorial shows a quick and easy way to select random numbers using PEEK and POKE to increase speed. The technique is also demonstrated as an alternative to the SOUND command.

When writing a BASIC program, it is often necessary to find the fastest possible method to achieve a desired result. When speed is important, a machine language subroutine is usually the best alternative. In many cases, however, using PEEK and POKE instructions instead of conventional routines can significantly increase the speed.

In each of the four example routines below, RAM location 540 is used as a timer. The term jiffy is used to denote 1/60 second. Location 540 counts backwards until it reaches zero. When the number 255 is POKed into this location, it will take 4¼ seconds to count back to zero.

Each routine begins with a GRAPHICS 0 command to clear the screen. You might want to try mode 2 later on to see how the elapsed time of each routine is affected. Standard text mode was chosen so the routines could be listed on the screen and the elapsed time displayed.

Time tests 1 and 2 show two ways to select a random number between 0 and 255. The first method is the conventional way. For demonstration purposes, the random number was selected ten times.

The second listing provides an alternative method which is four times faster. Our number is selected with a PEEK at location 20. This is also a jiffy counter, but unlike location 540, this one counts forward until it reaches 255. It is then reset to 0 and continues counting normally. This method of selection is only useful when a single random number is required. For example, to return a decision on a 50 percent probability, check location 20 for less than, or for equal to, 127. This method would not be effective if more than one number is needed within a short period of time.

It is, however, an excellent alternative in most cases, and is much faster than the conventional method because the multiplication is eliminated.

To obtain a truly random integer between 0 and 255, PEEK location 53770. Try the following one-line program to see the random number generator in action:

```
10 ? PEEK(53770):GOTO 10
```

Time test routines 3 and 4 loop through the 256 pitches of Atari's undistorted sound. Test 3 uses the conventional SOUND command. The execution time was 123 jiffies, or just over two seconds. Test 4 uses the POKE command. The difference was 17/60 second.

There are many situations where the PEEK and POKE commands can be used to speed up your BASIC programs. There are also things that could not be done at all in Atari BASIC were it not for PEEK and POKE.

Atari BASIC Time Test 1

```
5 GRAPHICS 0:LIST
10 POKE 540,255:FOR TEST=1 TO 10:X=RND(0)*25
  6:NEXT TEST:TIME=PEEK(540)
20 PRINT :PRINT "TIME=";255-TIME;" 60th of a
  second."
```

TIME=16 60ths of a second

Atari BASIC Time Test 2

```
5 GRAPHICS 0:LIST
10 POKE 540,255:FOR TEST=1 TO 10:X=PEEK(20):
  NEXT TEST:TIME=PEEK(540)
20 PRINT :PRINT "TIME=";255-TIME;" 60th of a
  second."
```

TIME=4 60ths of a second

Atari BASIC Time Test 3

```
5 GRAPHICS 0:LIST
10 POKE 540,255:FOR TEST=0 TO 255:SOUND 0,TE
  ST,10,2:NEXT TEST:TIME=PEEK(540)
20 PRINT :PRINT "TIME=";255-TIME;" 60th of a
  second."
```

TIME=123 60ths of a second

Atari BASIC Time Test 4

```
5 GRAPHICS 0:LIST :SOUND 0,0,0,0:POKE 53761,
  162
10 POKE 540,255:FOR TEST=0 TO 255:POKE 53760
  ,TEST:NEXT TEST:TIME=PEEK(540)
20 PRINT :PRINT "TIME=";255-TIME;" 60th of a
  second."

TIME=106 60ths of a second
```

7

Beyond Basic

7

1200 Memory Map: An Initial Examination

Ian Chadwick

Although a short-lived product on the commercial market, the Atari 1200XL managed to make it into quite a few homes before the line was dropped to make way for the new line. Not that the 1200 was a bad product; it simply lacked several competitive features, such as expansion capability.

Compatibility with software written for 400 and 800 machines is possible only if the programs obeyed the rigid restrictions of the official operating system routines, laid out in the Atari technical manuals. Much software makes direct jumps into the OS that cause programs to crash when run on the 1200. BASIC programs usually work, but there may be difficulty with PEEK, POKE, and USR routines.

The following material is all taken from official Atari releases, including the technical notes for the 1200XL. Memory locations can be cross-referenced with the description in *COMPUTE!'s Mapping the Atari* when they are described as *moved*. This is the location these routines or locations have been moved to in the 1200, but they still perform the same function as in the 400 or 800. I have tried to provide all known ranges of values and proper explanations, usually taken from the rare 1200XL technical manual but not available in most outlets. I suggest that you try POKEing different values in these locations to see the results.

The format attempts to follow that of *Mapping the Atari* as closely as possible. References to 400/800 memory use relate directly to the Revision B ROMS, not always earlier versions. I trust it will prove a useful guideline for 1200 owners.

DECIMAL	HEX	LABEL
00	00	LNFLG
Reserved for in-house debugging routines. 400/800 use: LINZBS; used in power-up sequence.		
01	01	NGFLAG
Reserved for power-up self-testing routines. 400/800 use: see location 00.		

28 1C ABUFPT
 Reserved for OS use, most likely as a buffer pointer.
 400/800 use: PTIMOT, moved to 788 (\$314).

29 1D ABUFPT
 Reserved for OS use.
 400/800 use: PBPNT, moved to 734 (\$2DE).

30 1E ABUFPT
 Reserved for OS use.
 400/800 use: PBUFSZ, moved to 735 (\$2DF).

31 1F ABUFPT
 Reserved for OS use.
 400/800 use: PTEMP, now deleted.

54 36 LTEMP
 Temporary buffer for loader routine. The technical notes contain extensive information about enhancements to the peripheral handling in the 1200. One inclusion is a relocating loader, used to upload peripheral handlers through the SIO. Of particular importance are the two additional device inquiries (polls) to the 1200XL. See the *1200XL Operating System Manual* for more information.
 400/800 use: CRETRY, moved to 668 (\$29C).

55 37 LTEMP
 Same as above.
 400/800 use: DRETRY, moved to 701 (\$2BD).

74 4A ZCHAIN
 Temporary storage for handler loader.
 400/800 use: CKEY, moved to 1001 (\$3E9).

75 4B ZCHAIN
 Same as above.
 400/800 use: CASSBT, moved. Official sources put this, as well as CKEY, above, at 1001. I suspect it is at 1002 (\$3EA) instead.

96 60 FKDEF
 Function key definition table pointer, low byte. You can redefine the function keys alone, by setting up an eight-byte table for the keys F1 to F4 and SHIFT F1 to SHIFT F4. You then assign each byte a value (the internal code: see "Reading the Keyboard Codes" and Appendix A) to correspond to the key. This way, you can get the function keys to act as any other keys. You must, however, make sure that you do not assign to the function keys their own value (138 to 141, \$8A to \$8D). That is, you must not

make F1 perform F1 (138, \$8A); otherwise you will generate an endless loop in which the system goes to check what the key should be, sees it is the same, returns, sees there is a table to check, goes back, etc. See locations 121, 122, (\$79, \$7A) for information on redefining the keyboard.

400/800 use: NEWROW, moved to 757 (\$2F5).

97 61 FKDEF

Same as above, high byte.

400/800 use: NEWCOL, moved to 758 (\$2F5).

98 62 PALNTS

Flag for PAL or NTSC version display handler. This was previously at 53268 (\$D014).

400/800 use: NEWCOL, second register, moved to 759 (\$2F6).

121 79 KEYDEF

Pointer to key definition, low byte. You can redefine almost the entire keyboard on the 1200XL by setting up a 192-byte table and POKEing the address in these two bytes. When you press a key, the system will respond with the new definition you have given it.

The table consists of three 64-byte portions: lowercase keys, SHIFT + key, CTRL + key. Each key corresponds to a byte as below:

DEC/HEX	KEY	DEC/HEX	KEY
00/00	L	17/11	HLP
01/01	J	18/12	C
02/02	;	19/13	F3
03/03	F1	20/14	F4
04/04	F2	21/15	B
05/05	K	22/16	X
06/06	+	23/17	Z
07/07	*	24/18	4
08/08	O	25/19	
09/09		26/1A	3
10/0A	P	27/1B	6
11/0B	U	28/1C	ESC
12/0C	RET	29/1D	5
13/0D	I	30/1E	2
14/0E	-	31/1F	1
15/0F	=	32/20	,
16/10	V	33/21	SPACE

DEC/HEX	KEY	DEC/HEX	KEY
34/22	.	49/31	
35/23	N	50/32	0
36/24		51/33	7
37/25	M	52/34	BACKS
38/26	/	53/35	8
39/27	logo key	54/36	<
40/28	R	55/37	>
41/29		56/38	F
42/2A	E	57/39	H
43/2B	Y	58/3A	D
44/2C	TAB	59/3B	
45/2D	T	60/3C	CAPS
46/2E	W	61/3D	G
47/2F	Q	62/3E	S
48/30	9	63/3F	A

Note that there are intentional blanks in the table where no key correspondence exists. Using the table above, to redefine the A key, you would change the 63rd byte in each of the three contiguous parts: the first to redefine the lowercase, the second for the SHIFTeD key, and the last for the CTRL and key.

You may place any value between 0 and 255 (\$FF) in these bytes; values between 0 and 127 (\$7F), 146 and 255 (\$92 to \$FF) are the ATASCII codes. The following values have special meanings to the 1200XL:

DEC/HEX	USE:
128/80	Ignored as invalid key combination.
129/81	Turns the keys to inverse output (normal becomes black on colored screen).
130/82	Upper/lowercase toggle.
131/83	Uppercase lock.
132/84	Control key lock.
133/85	End of file.
134/86 to 136/88	are ATASCII code.
137/89	Toggles keyboard click on or off.
138/8A	Function one; that use defined by the function key description.
139/8B to 141/8D	are functions two, three, and four, respectively.
142/8E	Cursor to home (upper-left corner of the screen).
143/8F	Cursor to bottom left-hand corner of the screen.
144/90	Cursor to the left margin, beginning of the physical line.

145/91 Cursor to the right margin, end of the physical line.

See locations 96, 97 (\$60, \$61) for redefining the function keys alone, without redefining the rest of the keyboard. You cannot redefine the following keys, since they are either hardwired into the system or operate as a special case:

BREAK, SHIFT, CTRL, OPTION, SELECT, START, RESET, HELP, CTRL-1, CTRL-F1 to CTRL-F4.

400/800 use: ROWINC, moved to 760 (\$2F8).

122 **7A** **KEYDEF**

Same as above, high byte.

400/800 use: COLINC, also called CLINC, moved to 761 (\$2F9).

563 **233** **LCOUNT**

Temporary counter for loader register. See section 5.0 in the *1200XL Operating System Manual* for information concerning the relocatable loader routine.

400/800 use: SPARE, not used.

568,569 **238,239** **RELADR**

Relocatable loader routine address pointers.

400/800 use: same as above.

581 **245** **RECLEN**

Loader routine variable register.

400/800 use: same as above.

583-618 **247-26A**

Reserved for future use.

400/800 use: LINBUF, now deleted from the OS.

619 **26B** **CHSALT**

Character set pointer, defines which character set is to be called into use at the next toggle of the CTRL-F4 keys. Initialized to 204 (\$CC) to point to the international set.

400/800 use: see location 583 (\$247).

620 **26C** **VSFLAG**

Fine scroll temporary register.

400/800 use: see location 583 (\$247).

621 **26D** **KEYDIS**

Keyboard disable register. POKE with 0 to enable keyboard use, 255 to disable it. Remember that you can reenale keyboard use from the keyboard by pressing CTRL + F1. You may also disable

the keyboard with the same combination. LED 1 will be on when the keyboard is disabled.

400/800 use: see location 583 (\$247).

622 26E FINE

Flag for fine scroll enable in GR. 0 (text) mode. POKE with 255 for fine scrolling, 0 for coarse scrolling. Follow this POKE with a GR.0 command or an OPEN command for device E:. The display list created for fine scrolling will be one byte larger than the normal, coarse scroll list. The OS also places the address of a DLI (display list interrupt) at VDSLST (512, 513; \$200, \$201). The color register at 53271 (\$D017) is also altered for the last visible line on the screen.

400/800 use: see location 583 (\$247).

648 288 HIBYTE

Register for loader routine.

400/800 use: CSTAT, deleted from OS use.

654 28E NEWADR

Loader routine register, same as above.

400/800 use: reserved (spare).

668 29C CRETRY

Moved from 54 (\$36).

400/800 use: TMPX1, now deleted.

701 2BD DRETRY

Moved from 55 (\$37).

400/800 use: HOLD5, now deleted.

713,714 2C9,2CA RUNADR

Register for loader routines.

400/800 use: spare.

715,716 2CB,2CC HIUSED

Same as above.

400/800 use: spare.

717,718 2CD,2CE ZHIUSE

Same as above.

400/800 use: spare.

719,720 2CF,2D0 GBYTEA

Same as above.

400/800 use: spare.

721,722 2D1,2D2 LOADAD

Same as above.

400/800 use: spare.

723,724 2D3,2D4 ZLOADA

Same as above.

400/800 use: spare.

725,726 2D5,2D6 DSCTLN

Disk sector size register. The 1200XL establishes sector size at 128 (\$80) bytes at power-up or reset, but you can alter the size to any length from 1 to 65536 (\$FFFF) bytes. You can also write to the disk without write-verify by using the command "P".

400/800 use: spare

727,728 2D7,2D8 ACMISR

Reserved, purpose unknown

400/800 use: spare.

729 2D9 KRPDEL

Keyboard auto-key delay rate; the time lapsed before the auto-key repeat begins. Default is 48. POKE with the number of VBLANK intervals before the repeat begins; each VBLANK is 1/60 of a second, so a value of 60 would equal a one-second delay.

400/800 use: spare.

730 2DA KEYREP

Keyboard auto-key rate. Default is six, which gives a rate of ten characters per second. POKE with the number of VBLANK intervals before a keystroke is repeated; at one, you will get 60 characters per second repeat rate! See the *1200XL Operating System Manual* for information concerning the difference between NTSC (North American) and PAL (English) system rates (NTSC has a 1/60 rate, PAL 1/50).

400/800 use: spare.

731 2DB NOCLIK

Key click disable; POKE with 255 to disable, 0 to enable. In the older machines, the only way to properly disable the click was to install an on/off switch. You may also use the CTRL-F3 keys to toggle keyboard click on and off.

400/800 use: spare.

732 2DC HELPFG

Flag for the HELP key enable. POKE with 0 to clear it. When PEEKed, 17 = HELP key pressed, 81 = SHIFT + HELP pressed, and 145 = CTRL + HELP pressed. HELPFG is *not* cleared after the HELP key has been pressed once. You must clear it yourself under program control.

400/800 use: spare.

733 2DD DMASAV

DMA state save register. This saves the screen graphics state when you disable the screen (CTRL-F2) for faster calculations.
400/800 use: spare.

734 2DE PBPNT

Moved from 29 (\$1D).
400/800 use: spare.

735 2DF PBUFSZ

Moved from 30 (\$1E).
400/800 use: spare.

745 2E9 HNDLOD

Loader routine handler flag.
400/800 use: spare.

746-749 2EA-2ED DVSTAT

These four device status registers are also used by the 1200XL to contain information sent back to the computer by the peripheral after a type three or four poll (these are new poll types; see the *1200XL Operating System Manual*). The bytes will contain, in order:
746: Low byte of the handler size, in bytes (must be an even number).

747: High byte of the handler size.

748: Device SIO (serial I/O) address to be used for loading.

749: Peripheral revision number.

756 2F4 CHBAS

Character set select, as in the 400/800. Default is 224 (\$E0) for domestic set; POKE with 204 (\$CC) for the international set. When you press CTRL-F4, the value in CHBAS is swapped with that in CHSALT (619; \$26B). If you want to select the international set for the next toggle, POKE 200 (\$C8) here, rather than 204 (\$CC). According to the *1200XL Operating System Manual*, the OS tests CHBAS and if it finds 200 in that location, swaps the value with that in CHSALT, usually 204. When the international character set is toggled, LED 2 is lit.

757 2F5 NEWROW

Moved from 96 (\$60).
400/800 use: spare.

758,759 2F6,2F7 NEWCOL

Moved from 97, 98 (\$61, \$62).
400/800 use: spare.

760 2F8 ROWINC

Moved from 121 (\$79).
400/800 use: spare.

761 2F9 COLINC

Moved from 122 (\$7A).
400/800 use: spare.

782 30E JMPERS

Option jumpers, designed to tell the OS how the system is configured. Only J1 (Bit 0) has been assigned. If Bit 0 equals zero (low), then the self-test will run. Bits 1-3 are reserved for future use, bits 4-7 are unused.

400/800 use: ADDCOR, deleted.

788 314 PTIMOT

Moved from 28 (\$1C).
400/800 use: TEMP2, moved to 787 (\$313).

829 33D PUPBT1

Power-up and reset register one.
400/800 use: reserved (spare).

830 33E PUPBT2

Power-up and reset register two.
400/800 use: reserved (spare).

831 33F PUPBT3

As above, register three.
400/800 use: reserved (spare).

1000 3E8 SUPERF

Screen editor register.
400/800 use: reserved (spare).

1001 3E9 CKEY

Moved from 74 (\$4A).
400/800 use: reserved (spare).

1002 3EA CASSBT

Moved from 75 (\$4B).
400/800 use: reserved (spare).

1003 3EB CARTCK

Cartridge checksum. Likely the way the system ascertains the size (8K or 16K) of a cartridge when in place.
400/800 use: reserved (spare).

1005-1016 3ED-3F8 ACMVAR

Reserved for OS variables. On power-up and coldstart, variables from 1005 to 1023 (\$3ED to \$3FF) are set to zero. On warmstart or reset, they are not changed.

400/800 use: reserved (spare).

1017 3F9 MINTLK

Same as above.

400/800 use: reserved (spare).

1018 3FA GINTLK

Cartridge interlock register.

400/800 use: reserved (spare).

1019,1020 3FB,3FC CHLINK

Handler chain.

400/800 use: reserved (spare).

1792-7419 700-1CFB

Used by DOS when loaded, otherwise available as user RAM.

39967-40959 9C1F-9FFF

Display list and screen RAM. This will get moved to lower addresses if the cartridge is 16K (using up the memory from 32768 to 49151; \$8000 to \$BFFF). The normal 8K cartridge uses RAM between 40960 and 49151 when installed (\$A000 to \$BFFF). Two control lines tell the system a cartridge is installed.

49152-52223 C000-CBFF

OS ROM. In the 400/800, the block from 49152 to 53247 (\$C000-\$CFFF) was unused and unusable. Many of the interrupt handler routines have been moved into this block now, the reason for the incompatibility with 400/800 programs which jump to the old locations rather than to official vectors in RAM.

The bytes between 49152 and 49163 (\$C000-\$C00B) contain identification and checksum data for the ROM between 49152 and 57343 (\$DFFF) using the following format:

DEC/HEX USE:

49152/C000 Checksum low byte; sum of all of the bytes in ROM except the checksum bytes themselves.

49153/C001 Checksum high byte.

49154/C002 Revision date, using the form DDMMYY, where each four bits is a BCD digit. The byte has two four-bit numbers for D1 and D2 in the upper and lower halves, respectively.

49155/C003 Revision date, month code, M1 and M2.

49156/C004	Revision date, year code, Y1 and Y2.
49157/C005	Option byte, reserved. Contains zero for the 1200XL.
49158/C006	Part number, using the format AANNNNNNN, where A is an ASCII character and N is a four bit BCD digit. This byte is A1.
49159/C007	Part number, A2.
49160/C008	Part number, N1 and N2.
49161/C009	Part number, N3 and N4.
49162/C00A	Part number, N5 and N6.
49163/C00B	Revision number.

52224-53247 CC00-CFFF CHARSET2

International character set, one of two in the 1200. The other is at the same place as in the 400/800; 57344-58367 (\$E000-\$E3FF).

53248-53503 D000-D0FF GTIA

GTIA and graphics registers, as in the 400/800. The self-test code is physically located between 53248 and 55295 (\$D000 to \$D7FF) but moved to 20400 to 22527 (\$5000 to \$57FF) when called up.

53504-53759 D100-D1FF

Unused in both 400/800 and 1200 versions.

53760-54015 D200-D2FF POKEY

POKEY registers, same as in the 400/800.

54016-54271 D300-D3FF PIA

PIA registers, same as in the 400/800.

54017 D301 PORTB

Used to control the LEDs and the memory management, enabling you to disable the OS ROM and enable the RAM. Bit 0 controls location 49152-53247 (\$C000-\$CFFF) and 55296-65535 (\$D800-\$FFFF). When set to zero, the OS is replaced by RAM. However, unless another OS has been provided, the system will crash at the next interrupt. Bit 7 controls the RAM region 20480-22527 (\$5000-\$57FF) and is normally enabled (set to one). If disabled (set to zero), then the OS ROM is enabled, the memory access remapped and access provided to the self-test code physically present at 53248-55295 (\$D000-\$D7FF). If LED 1 is on, then the keyboard is disabled. If LED 2 is on, then the international character set is selected.

400/800 use: PIA PORTB. Since there are only two controller jacks (PORTA), this is no longer used in the 1200, meaning only two game controllers may be attached at once, rather than four.

54272-54527 D400-D4FF ANTIC

ANTIC registers, same as in 400/800.

54528-55295 D500-D7FF

Unused in both 400/800 and 1200 versions of the OS. Any access read or write in the 54528 to 54783 (\$D500 to \$D5FF) range enables the cartridge control line CCNTL in the cartridge interface as in the 400/800.

55296-57343 D800-DFFF FP

Floating point package as in the 400/800. The 1200XL corrects a bug in the FP package which was in the REV B ROMs. You now get an error status when you try to calculate the LOG or LOG10 of zero.

57344-58367 E000-E3FF CHARSET1

Domestic character set, as in the 400/800. The international character set location is listed above. This is the default set. Register 756 (\$2F4) defines which is in use (see above).

58368-65535 E400-FFFF OS

OS ROMS. There are many changes in the 1200 OS, making it quite different from the 400/800 OS, but advertised entry points and vectors have been left the same. There are five new fixed entry point vectors which have been added to the 1200XL:

58496/E480 JMP PUPDIS: entry to power-on display.

58499/E483 JMP SLFTST: entry to the self-test code.

58502/E486 JMP PHENTR: entry to the handler, uploaded from peripheral or disk.

58505/E489 JMP PHULNK: entry to uploaded handler unlink.

58508/E48C JMP PHINIS: entry to uploaded handler initialization.

58481 E471

The Atari 400/800 had a blackboard mode; the Memo Pad mode you saw when typing BYE in BASIC. This no longer exists on the 1200XL; it has been replaced by the noninteractive Atari advertisement logo.

Bytes from 65518 to 65529 (\$FFEE to \$FFF9) contain checksum and identification for the ROM block 57344 to 65535 (\$E000 to \$FFFF) in a similar format to that at location 49152 (\$C000). The bytes used are as follows:

DEC/HEX	USE
---------	-----

65518/FFEE	Revision date D1 and D2.
-------------------	--------------------------

65519/FFEF	Revision date M1 and M2.
-------------------	--------------------------

65520/FFF0	Revision date Y1 and Y2.
65521/FFF1	Option byte; hardware product identifier; for the 1200XL it should read one.
65522/FFF2 to 65526/FFF6	Part number using the form AANNNNNN.
65527/FFF7	Revision number.
65528/FFF8	Checksum byte, low byte.
65529/FFF9	Checksum byte, high byte.

Bytes from 65530 to 65535 (\$FFFA to \$FFFF) contain power-on, RESET, NM, and IRQ vectors.

65521 FFF1

If you PEEK here, you should get one and then 65527 (\$FFF7) will have the revision number. If not one, then the product code will be here and 65527 will contain the OS revision number. This identifies the OS as that of the 1200XL. Accordingly, if you PEEK 65527 and 65528 (\$FFF7, \$FFF8) and get 221 (\$DD) and 87 (\$57) respectively, you have the 400/800 Revision A ROMS. If you get 243 (\$F3) and 230 (\$E6), you have the Revision B ROMS. PAL versions will read 214 (\$D6) and 87 (\$57), 34 (\$22) and 88 (\$58) respectively. If location 64728 (\$FCD8) is not 162 (\$A2) then the product is a 1200XL or future computer.

New Graphics Modes

Four new graphics modes are available on the 1200 from BASIC: GRAPHICS 12, 13, 14, and 15. These are the same as modes described in the technical manuals but previously unavailable in BASIC.

GRAPHICS 12 is ANTIC mode 4, a four-color mode (plus background). Each character on the screen is the same size as a GRAPHICS 0 character but only four pixels are displayed instead of eight as in GRAPHICS 0. It can be well used by a redefined character set. The screen has 20 lines; to obtain the full 24 lines, use GRAPHICS 12 + 16.

GRAPHICS 13 is ANTIC mode 5, another four-color mode (plus background), this time with characters double the physical space of the GRAPHICS 0 characters. As in GRAPHICS 12, only four pixels are displayed; the system interprets definition in the character sets by bit pairs, rather than single bits as in GRAPHICS 0. The screen has ten lines and can be expanded to 12 by GRAPHICS 13 + 16. Both GRAPHICS 12 and GRAPHICS 13 use 40 bytes of screen RAM per line.

In both GRAPHICS 12 and GRAPHICS 13, the color of the screen pixel depends on the bit pair in the byte addressed. Each character can be built of eight bytes like the GRAPHICS 0 characters, but bits are paired for screen presentation. If the bits have the value below, then the color shown appears on the screen:

VALUE/BINARY**COLOR****0/00****BAK****1/01****PF0****2/10****PF1****3/11**

If Bit 7 of the character = 0 (the color modifier), then PF2 is used, else if Bit 7 = 1, then PF3 is used.

GRAPHICS 14 is ANTIC mode 12 (\$C), a two-color mode with a resolution of 160 pixels wide by 192 pixels high. This is sometimes called GRAPHICS "6½" because each line is one scan line high where GRAPHICS 6 is two scan lines high. Colors used are BAK and PF0. Only the first bit of a screen byte is used to identify the color.

GRAPHICS 15 is ANTIC mode 14 (\$E), known as GRAPHICS "7½" and used in many popular commercial programs such as Datasoft's *Micropainter*. It is a four-color mode with a resolution of 160 across by 192 down, each mode line being one scan line high. Colors used are BAK and PF0 to PF2. Only the first two bits in a screen byte are used to identify the color of the byte.

Data for New Screen Modes

Mode	Horizontal Line	Vertical Line	Colors	Memory Used:	
				Split Screen	Full Screen
12	40	20/24	5	1154	1152
13	40	10/12	5	664	660
14	160	160/192	2	4270	4296
15	160	160/192	4	8112	8138

Final Notes

If you have a copy of *Mapping the Atari*, you may find it useful to make a note in the margins of the new locations of interrupt and other routines as defined by the vectors. Most of these are located between 512 and 1151 (\$200 to \$47F). These new pointers will show you where routines have been moved in the 1200.

A small one-pixel shift in the 1200's display may cause some programs to show different colors (particularly artifact colors in GRAPHICS 8) than they do on the 400/800. Colors (but not graphics modes) now conform to those displayed by the earlier CTIA chip.

Some Revision B enhancements which are also in the 1200XL should be mentioned. First, the display handler will not clear memory beyond that indicated by RAMTOP (location 106; \$6A). This means you can store data or machine language routines above the graphic display and have them remain intact when changing graphics modes. Second, you can assign a printer number from P1 up to P8. The printer handler inserts an EOL in the printer buffer if none is there, before sending the buffer to the printer on a CLOSE. This allows the printer to immediately print the last line, rather than having to force it to do so. The CIO places an EOL in the input buffer when a record longer than the buffer size is being read. This allows you to still read a portion of a record even if a large enough buffer was not provided. Finally, the screen clear code will work no matter what the cursor coordinates are.

If at all possible, try to obtain a copy of the *1200XL Operating System Manual*. Much of what is vague here is explained there. There are many other, more subtle and technical differences between 400/800 use and 1200XL use. These are best explained in Atari's own manuals. The manual also contains instructions on how to redefine the Atari keyboard as a Dvorak layout and define GRAPHICS 12 and 13 characters, and it gives specific information on the new peripheral poll types and their use.

Merging Machine Language into BASIC

Fred Pinho

Merging machine language subroutines can be a time-consuming task. The program offered here will allow you to add machine language to a BASIC program as a string or as DATA statements.

You've just bought your Assembler Editor cartridge, and you're starting to get into machine language programming. Hold it, before you go any further. If you haven't already heard, your assembler manual is chock full of errors. Run, don't walk, to the Atari hot line to request their errata sheets. It will save you grief and headaches, especially if you are cassette dependent.

After writing and debugging your first machine language program with your Assembler Editor cartridge, you can now save it to cassette or disk as a binary file. You can also load it back into the computer and run the machine language program directly. But what if you want to combine this routine with a BASIC program? This is the objective of a majority of beginning machine language programmers. If you look on pages 66-67 of the Assembler Editor manual, you will find a merger program. However, the program is clumsy and unwieldy, especially in its handling of problem code values (such as the one which is the ATASCII equivalent of quotation marks).

To overcome this problem, I've provided Program 1. This program will take your machine language and automatically convert it into a complete BASIC subroutine. This can then easily be added to your BASIC program. The subroutine is complete within itself. It requires only:

- That your program have line numbers no greater than 31000.
- That you call the subroutine as early as possible in your program.

This will allow you to reuse the subroutine variables in your program if you wish. Also the DIMension statements will be declared at the start of the program.

Your Options

This utility program has a great deal of flexibility built-in. You can choose to store your machine language in a variety of ways:

- As strings (probably the safest and most versatile method). The program will automatically generate the strings plus the DIMension statements to support them. It also will take care of the troublesome codes of 34 (ATASCII for quotes) and 155 (ATASCII for RETURN).
- Storage at a specific location in memory. The location can be the same as specified in your binary file or it can be changed. The program will then generate a series of DATA statements. It will also provide a short routine that will READ the data and POKE it into memory.
- Any number and combination of string and location storage can be used. The program will combine them into a single subroutine to set them up all at once. Just merge with your BASIC program and add a GOSUB to this subroutine.
- The program will check your keyboard input and prompt you when you've made an error.

The program, as written, sits in slightly less than 5K bytes of RAM after DIMensioning of arrays and strings. I've run it through the "Masher" program from the Atari Program Exchange. However, this saves only about 500 bytes. The program also then becomes very difficult to follow. So I've kept it as is. Type the program in and LIST it to disk or cassette. Don't SAVE it.

Using the Utility

To use the utility program, first store your machine language to disk or cassette as a binary file. If your source program is in RAM, this can be done through the assembler with this command: `ASM,,#D:<filename>` for disk or `ASM,,#C:` for cassette.

Note that what you wrote was the source code, not the actual object code which is the machine language program. Once you've done the above, turn off the computer to wipe out the source program. Then remove the Assembler cartridge, insert the BASIC cartridge, and boot the DOS into memory.

If your program has already been assembled (converted to machine language) and the final machine language resides in RAM, then do the following:

For Disk

SAVE #Disk File <starting address><end of routine address

Example: SAVE #D:PROGRAM.OBJ<1400<17FF

For Cassette

SAVE #C:<start address><end address

Note that all addresses are in a hexadecimal.

Again, shut off the computer and replace the Assembler with the BASIC cartridge.

To use this utility both the utility and the machine language program must be in RAM. The utility program occupies about 5K bytes of memory. Thus you must be careful to locate your machine language program so that it does not interfere with the BASIC program. You can locate the machine language either in page 6 or high up in memory just below the display list. To help you with the second method, the tables below define usable and safe living space for your machine language program.

Table 1. Disk-Based System

Computer RAM Installed	Suggested Safe Memory			
	Decimal		Hexadecimal	
	From	To	From	To
8K	Not enough memory			
16K	12750	15390	31CE	3C1E
24K	12750	23582	31CE	5C1E
32K	12750	31774	31CE	7C1E
40K	12750	39966	31CE	9C1E
48K	12750	39966	31CE	9C1E
Note: Assumes that you are in GRAPHICS 0, that the BASIC cartridge is installed, and that the first part of DOS 2.0S (mini-DOS) is loaded. The mini-DOS occupies 5628 bytes.				

Table 2. Cassette-Based System

Computer RAM Installed	Suggested Safe Memory			
	Decimal		Hexadecimal	
	From	To	From	To
8K	7100	7198	1BBC	1C1E
16K	7100	15390	1BBC	3C1E
24K	7100	23582	1BBC	5C1E
32K	7100	31774	1BBC	7C1E
40K	7100	39966	1BBC	9C1E
48K	7100	39966	1BBC	9C1E

Note: Assumes that you are in GRAPHICS 0 and that the BASIC cartridge is installed.

Machine Language to BASIC

To convert your machine language to BASIC, proceed as follows:

1. Load your machine language subroutine into its *safe* area. If from disk, first load the second part of DOS and then use option L (Binary Load). Then go back to BASIC. If you have a cassette, be careful. Page 65 of the Assembler Editor manual tells you to CLOAD your machine language. Trying that can give you a headache. The errata sheets from Atari give you a routine for cassette loading.
2. ENTER the utility program which has previously been LISTed to disk or cassette (see step 6 if you are using cassette).
3. RUN the program. The program will ask for the starting and ending addresses of your machine language routine in RAM. Answer in *decimal* only! All keyboard inputs for this program must be in decimal form.
4. The program will then ask which method you desire for storage of your machine language. If you wish string storage, you will be prompted for the string name. You will also be asked if you wish a printout of the data to be inserted into the string. If so, you will be prompted to turn on the printer.
5. If you desire to store machine language at a specific location, you will be asked if you wish storage at the same memory location as specified in step 3. Alternatively, you can store it at a different location.

6. Finally you'll be asked if you wish to make any additional conversions. If yes, the program will loop back. If not, the computer will CLOSE all files and END. Your BASIC subroutine will be stored on disk as a file labelled MLR.LST. If you are using a cassette, see Program 2 for required program modifications.
7. After you're done, erase the utility program via NEW. Now enter your BASIC program. Finally, merge your machine language into your program by:

For Disk

ENTER "D:MLR.LST"

For Cassette:

See Program 2

8. Now that the two programs are merged, type in a GOSUB statement to reference the first line number of MLR.LST (or the equivalent cassette file).

And that's it; you're ready to go.

Storing Machine Language in Strings

I'd like to make some comments on storage of machine language in a string format. First, to do it correctly, you must write routines which are relocatable. That is, they must not contain any JMP or JSR instructions to a specific memory location within the program. Since the string can be located nearly anywhere in memory, nonrelocatable code will almost surely crash the computer. It's best to store your subroutines and data tables in page 6 of memory. These permanent addresses can then be safely called from within your routine.

Another problem lies in proofreading your string. If you load your data into a string and then PRINT it to the screen, you will see many weird and wondrous things. What is happening is that the screen editor is interpreting the function of the printed graphics symbols and carrying out the function. For example, if the graphics symbol in your string is that for a "delete character," the computer will slavishly do it. Thus the string symbols seen on your screen are *not correct* (unless you're lucky). To check your string, use the following routine in direct mode. (First RUN your program to DIMension and initialize your string):

```
L=LEN(string name$):FOR X=1 TO L: ?AS
C(string name$(X,X)); ", ";:NEXT X
```

This routine prints the actual value of each byte stored in the string.

Another serious problem with string storage of data is the occurrence of values of 34 or 155. The value 34 is the ATASCII representation of quotation marks. The value 155 is the ATASCII for RETURN. The presence of either will cause the screen editor to prematurely truncate your string and give you an error message. Thus the program does the following when it encounters either value:

1. It inserts a space character in the string and notes the position in the string.
2. It then writes the BASIC subroutine statements so that the values are inserted into the string without going through the screen editor. It uses the CHR\$ function for this purpose.
3. As presently set up, the program can handle up to 15 values of the quotes and of the RETURN characters. It checks for the total occurrence of these and warns you if there are too many.

There you have it. I hope this program makes the difficult world of machine language a little more enjoyable.

Table 3. Variables in Program 1

A\$	Used to receive yes or no responses
BATOP	Top memory location of utility program
D0\$	Holds name of string used to store machine language (ML)
F	Flag. Zero if string storage requested. Set to one if storage at a specific address is requested
I,S,T,X,Y	Loop counters
L	Length of string required to store ML
LS,LF	Initial and final position in string to be filled with data
LN	Line number of subroutine to be written for string storage
LNO	Line number of DATA statements to be written for ML storage at a specific address
LR	Remaining length of string after subtracting 80
N	Input value for choice of ML storage
QT	Total number of values of 34 in ML
QUOTE()	Holds position in string of ATASCII values of 34
RT	Total number of values of 155 in ML
RETRN()	Holds position in string of ATASCII values of 155
S	Temporary value for ML address
SE,FF	New starting and final address of ML
SO,FO	Initial starting and final address of ML

- V Counter to indicate number of routines to be stored at specific addresses
 W Indicates cell in array RETRN()
 Z Indicates cell in array QUOTE()

Program 1. Merging ML into BASIC Disk Version

```

10 CLR : GRAPHICS 0: POKE 752,1: POKE 756,209: ?
   "{5 SPACES} MACHINE LANGUAGE CONVERTER": G
   OSUB 600: POKE 756,224
20 DIM A$(1), D0$(3), QUOTE(14), RETRN(14)
30 D0$="{3 SPACES}": TRAP 580: GOSUB 740: V=0: O
   PEN #3,8,0, "D:MLR.LST": LNO=32050: LN=31000
   : F=0
40 ? : ? : ? "INPUT STARTING ADDRESS OF CODE":
   POKE 752,0: GOSUB 590: INPUT S: SO=S: SF=S
50 ? "INPUT FINAL ADDRESS OF CODE": GOSUB 590
   : INPUT S: FO=S: FF=S: GOSUB 640
60 ? "STORAGE METHOD FOR ROUTINE?": ? "
   {3 SPACES} 1. AT SAME ADDRESS": ? "
   {3 SPACES} 2. WITHIN A STRING?"
70 ? "{3 SPACES} 3. AT NEW ADDRESS?"
80 GOSUB 590: ? : ? "PLEASE TYPE NUMBER PLUS R
   ETURN!": INPUT N
90 IF (N<>1 AND N<>2 AND N<>3) THEN ? "
   {BELL}WRONG RESPONSE! TRY AGAIN!": GOSUB 60
   0: GOTO 60
100 IF N=3 THEN ? : ? "NEW STARTING ADDRESS F
   OR ROUTINE?": GOSUB 590: INPUT S: SF=S
110 IF N=3 THEN ? "NEW FINAL ADDRESS FOR ROU
   TINE!": GOSUB 590: INPUT S: FF=S
120 IF N=3 THEN IF FF-SF<>FO-SO THEN ? "
   {BELL}INCORRECT FINAL ADDRESS! TRY AGAIN
   !": ? : GOTO 110
130 IF N=1 OR N=3 THEN F=1: V=V+1: GOTO 180
140 L=FO-SO+1: GOSUB 680: GOSUB 610
150 ? "DO YOU WISH AN ASCII PRINTOUT": ? "OF
   YOUR STRING DATA!": GOSUB 590: INPUT A$
160 IF A$="Y" THEN N=4: ? "HIT RETURN WHEN TH
   E PRINTER IS ON!": GOSUB 590: INPUT A$: OPE
   N #2,8,0, "P:"
170 GOTO 260
180 ? #3; LNO; " DATA "; SF; ", "; FF: LNO=LNO+10
190 ? #3; LNO; " DATA ";
200 FOR I=0 TO 19
210 IF SO+I=FO+1 THEN POP : IF I THEN ? #3; ",
   "; -1: ? #3: LNO=LNO+10: GOTO 490
215 IF SO+I=FO+1 THEN IF I=0 THEN ? #3; -1: ?
   #3: LNO=LNO+10: GOTO 490

```

```

220 IF I THEN ? #3; ", ";
230 ? #3; PEEK(SO+I);
240 NEXT I: ? #3; LN=LN+10: SO=SO+20: GOTO 190
260 IF N=4 THEN ? #2: ? #2; "***DATA FOR "; D0$;
    "***"
270 LS=1: Z=0: W=0: ? #3; LN; " DIM "; D0$; "("; L; "
    ):";
280 IF N=4 THEN FOR I=0 TO L-1: ? #2; PEEK(SO+
    I); : IF I<L-1 THEN ? #2; ", ";
290 IF N=4 THEN NEXT I
300 LR=L-80: IF LR<=0 THEN LF=LS+L-1
310 IF LR>0 THEN LF=LS+80-1: L=LR
320 ? #3; D0$; "("; LS; ", "; LF; ")"=""; : ? #3; CHR$(3
    4); : FOR I=LS TO LF
330 IF PEEK(SO+I-1)=34 THEN ? #3; " "; : QUOTE(
    Z)=I: Z=Z+1: GOTO 360
340 IF PEEK(SO+I-1)=155 THEN ? #3; " "; : RETRN
    (W)=I: W=W+1: GOTO 360
350 ? #3; CHR$(PEEK(SO+I-1));
360 NEXT I: IF LR>0 THEN LS=LS+80: ? #3; CHR$(3
    4): ? #3; LN=LN+10: ? #3; LN; " "; : GOTO 300
370 ? #3; CHR$(34): ? #3; LN=LN+10
380 QT=0: RT=0: FOR X=0 TO 14: IF QUOTE(X) THEN
    QT=QT+1
390 IF RETRN(X) THEN RT=RT+1
400 NEXT X: IF QT=0 AND RT=0 THEN 490
410 ? #3; LN; "RESTORE "; LN+20: LN=LN+10
420 IF QT THEN ? #3; LN; " FOR X=1 TO "; QT; ": R
    EAD Z: "; D0$; "(Z,Z)=CHR$(34): NEXT X": LN=L
    N+10
430 IF QT THEN ? #3; LN; " DATA "; : FOR Y=0 TO
    QT-1: ? #3; QUOTE(Y); : IF Y AND Y<QT-1 THEN
    ? #3; ", ";
440 IF QT THEN NEXT Y: ? #3; LN=LN+10
450 IF RT THEN ? #3; LN; " FOR X=1 TO "; RT; ": R
    EAD Z: "; D0$; "(Z,Z)=CHR$(155): NEXT X": LN=
    LN+10
460 IF RT THEN ? #3; LN; " DATA "; : FOR Y=0 TO
    RT-1: ? #3; RETRN(Y); : IF Y AND Y<RT-1 THEN
    ? #3; ", ";
470 IF RT THEN NEXT Y: ? #3; LN=LN+10
490 GOSUB 740: ? "ALL DONE": GOSUB 590: INPUT A
    $
500 IF A$="N" THEN D0$="{3 SPACES}": CLOSE #2
    : GOTO 40
510 IF F=0 THEN 570
520 ? #3; "32000 W=0: V="; V; ": RESTORE 32050"
530 ? #3; "32010 READ X,Y: FOR I=X TO Y: READ Z
    : POKE I,Z: NEXT I"

```

```

540 ? #3;"32020 READ Z:IF Z<>-1 THEN ?";CHR$(34);"ERROR IN CODE! CHECK DATA STATEMENTS!";CHR$(34);":END"
550 ? #3;"32030 W=W+1:IF W<V THEN 32010"
560 ? #3;"32040 RETURN"
570 CLOSE #2:CLOSE #3:END
580 CLOSE #2:CLOSE #3:TRAP 40000:? "{BELL}ERROR ";PEEK(195); " AT LINE ";PEEK(186)+256*PEEK(187);"!":END
590 FOR T=10 TO 6 STEP -1:FOR S=8 TO 0 STEP -1:SOUND 0,15-S,10,T:NEXT S:NEXT T:SOUND 0,0,0,0:RETURN
600 FOR T=1 TO 400:NEXT T:RETURN
610 ? :? "INPUT TWO CHARACTER STRING NAME":? "PLUS THE $":GOSUB 590:INPUT D0$
615 IF LEN(D0$)<3 THEN GOSUB 750:GOTO 610
620 IF ASC(D0$(1,1))>90 OR ASC(D0$(1,1))<65 OR D0$(2,2)="$" OR D0$(3,3)<>"$" THEN GOSUB 750:GOTO 610
630 RETURN
640 IF SO<1792 THEN RETURN
645 IF FO>(256*PEEK(106)-1000) THEN ? "{BELL}I DON'T HAVE THAT MUCH MEMORY!":POP:GOTO 40
650 BATOP=PEEK(144)+256*PEEK(145):IF BATOP>50-100 THEN ? "{BELL}CAUTION! THIS PROGRAM MAY HAVE":GOTO 670
660 RETURN
670 ? "OVERRUN YOUR CODE! CHECK YOUR RESULTS!":GOSUB 600:RETURN
680 QT=0:RT=0:FOR I=0 TO L-1:IF PEEK(SO+I)=34 THEN QT=QT+1
690 IF PEEK(SO+I)=155 THEN RT=RT+1
700 NEXT I:IF RT<16 AND QT<16 THEN RETURN
710 ? "{BELL}WARNING! YOUR CODE CONTAINS":? "MORE THAN 15 ASCII VALUES FOR":? "RETURN OR QUOTES!"
720 ? "THUS I CANNOT PROCESS IT!":? "PLEASE MAKE ANOTHER CHOICE!":GOSUB 600:GOSUB 600
730 POP:GOTO 60
740 FOR I=0 TO 14:QUOTE(I)=0:RETRN(I)=0:NEXT I:RETURN
750 ? "{BELL}WRONG RESPONSE! TRY AGAIN!":RETURN

```

Program 2. Changes for Cassette Users (see notes below)

```
30 D0$="{3 SPACES}":TRAP 580:GOSUB 740:V=0:L  
   NO=32050:LN=31000:F=0  
35 OPEN #3,8,0,"C":? #3;"1 DATA ";;FOR I=0  
   TO 59:? #3;"0,";;NEXT I:? #3;"0";:? #3
```

Note: Line 35 writes a dummy line of DATA. This is needed because of a bug in the operating system. After the cassette handler is OPENed, the cassette motor will not stop running until a record is written to it. RUN the program and record the subroutine on tape. Then, before you enter your BASIC program, ENTER the subroutine from cassette. DELETE line 1. Then ENTER your BASIC program. Now type in GOSUB to the utility subroutine and you're ready to go.

Machine Language Sort Utility

Ronald and Lynn Marcuse

Machine language sorts are fast. With this sort utility you will be able to sort fixed or variable length records. These programs will not run on the 800XL.

There have been occasional articles in the various personal computer magazines concerning the sorting of data files. Some of these have presented sort routines written in BASIC that can be used in existing programs. The complex string handling required by the sort logic is not really suitable for BASIC's rather slow execution speed. Clearly, any type of repetitive string manipulations as performed by sorting or searching functions would definitely benefit from machine language. If you continue reading you will find out how much faster machine language really is.

Before we get into the programs themselves, it would probably be beneficial to include some background information. The verb *to sort* is defined as: "to put in a certain place or rank according to kind, class or nature; to arrange according to characteristics." This comes pretty close to what we sometimes want to do with the data we store in our computers and files: put it in some kind of order. Once we have arranged it we can search it quicker (imagine a disorganized phone book), list it in a more readable format, or even match it to other files that have been sorted the same way.

The Main Questions

First we must decide where will we do the actual sorting. All of us have arranged things on a desk or table. Our sort area is, therefore, the desk or table that we use. In a computer system we have a choice of using the memory within the machine (internal) or our disk drive (external). There are problems with both of these. Computer memory is limited in size and this, in turn, limits the number of records that can be read in. The disk drive may be able to hold more data, but the speed of the device is snail-like when compared to memory. We can also use both. Divide the file up

into smaller chunks which can be sorted in memory, store these on disk as temporary files, and then merge all of them together. This process is usually referred to as sub-listing or sort-merge.

The next question involves the type of sort logic (there are many ways of putting things in order). The algorithm used here is called a *bubble sort*. The file or list is examined two records at a time. If the second has a lower sort key than the first, the two will exchange places within the file. Why then, you ask, is it called a bubble sort: because records appear to bubble upward in memory (I didn't coin the phrase). Although this is not a very exotic methodology, it does offer several advantages such as requiring no other memory allocations for sorting and a rather quick speed if the file is not too disorganized. It will also not disturb the relative positioning of records that have equal sort keys.

There are numerous other types of sort algorithms. A *selection sort* would go through a list of (n) items ($n-1$) times, pulling out the next lowest record and adding it to the current end of a new list. This would need double the memory though. A *selection and exchange sort* would perform a similar function within the main sort area, selecting the lowest element during each pass, moving it upward in the list to be exchanged with the element occupying its new position. This method tends to upset the existing relative positioning. Other types involve binary tree searches and more complex algorithms.

The difference between fixed and variable length records is really just that. Fixed length records are all exactly the same size, while variable implies that all or many of the records in the file may vary in length. Record 1 may be 80 bytes long, record 2 may be 120, etc.

Why Machine Language

The choice of language is, as stated above, rather clear. Unless you have a lot of time to kill, it must be in machine language. When you're doing several hundred thousand (or million) character comparisons and swaps, you don't have time to pull out a BASIC/machine language dictionary for each line in the program (this, in essence, is what the BASIC interpreter does).

Here are some representative execution times, based on some testing we did a while back. The speeds are approximate and do not include disk input/output time. The test file consisted of 200 records, each 75 characters in length. The sort key occupied ten positions:

BASIC selection/exchange sort (in memory) - 8 minutes
BASIC bubble sort (in memory) - 12 minutes
BASIC selection sort (on disk) - 2 hours plus (hit BREAK key)
Machine language bubble (memory) - 3 seconds

The sort program was developed with flexibility in mind. It will sort fixed length or variable length records from 2 through 250 bytes in length. The sort key itself may be located anywhere in the record and can be any length (up to the size of the record). It will sort in either ascending or descending order. The records themselves must be comprised of ATASCII characters. While in memory, they need not be terminated by end-of-line (\$9B) characters.

The nominal limit of 250 characters is imposed by a possible bug in Atari's DOS II. The second half of page 5 (memory addresses 0580-05FF hex, 1408-1535 decimal) appears to be utilized as an internal I/O buffer. When more than 128 bytes are input, the excess winds up on page 6. The sort program also resides in the safe user area of page 6 (beginning at \$0680 or 1664). There is a physical law that states two things cannot occupy the same place at the same time. This also holds true in computer memory. The program has been pushed as far into page 6 as it can go.

Using the Sort

In order to use the sort, you must feed it certain parameters. The record length must be POKed into location 205 (\$00CD). The sort type (0-Ascending, 1-Descending) would be POKed into 206 (\$00CE). The starting and ending positions of the sort key will also have to be POKed into locations 203 (\$00CB) and 204 (\$00CC). The program is expecting to see the offset of the sort key. The offset is the number of positions in front of that byte. For example, the first position of a record has a 0 offset, the second has an offset of 1, and the hundredth has an offset of 99. The USER function that calls the sort will also pass the address of the string containing the file and the record count. For those who are a little unsure of what this is all about, there are a few examples coming up.

Now that you have a routine that will sort your data faster than you can say Rumpelstiltskin, how do you use it? Here are several suggestions. The easiest method is to link through our sort/file loader in Program 1 (fixed length only). Your existing program that is processing the data file is probably much, much

longer than the short loader. The main advantage of using a small program is that you wind up with more free memory. And, since memory is our sort area, the more that is free, the larger the file. If you don't type the REMark statements, you'll have an even larger sort area. The disk file must be fixed length records terminated by end-of-line characters. Your existing processing program must contain the POKEs mentioned above. It may look something like this:

```
POKE 203,SKEYA-1:POKE 204,SKEYB-1:PO
KE 205,RECLN:POKE 206,0 (for Ascend
ing)
```

The call to the loader would be a RUN "D:SORTLOAD" (give the loader this filename when you save it). The sort/file loader must have your filename in the variable F\$ and your program name in P\$. If your processing program handles several files, you can also pass the filename by using the following statements. First, your program:

```
FOR I=0 TO 14:POKE 1640+I,32:NEXT I
FOR I=0 TO LEN(F$):POKE 1640+I,ASC(
$(I,I)):NEXT I
```

Note: F\$ is your files name

The sort/file loader will require the following lines to be added:

```
70 FOR I=0 TO 14:F$(I,I)=CHR$(PEEK(1
640+I)):NEXT I
80 IF F$(1,2)<>"D:" THEN ? "ERROR":E
ND
```

If your processing program or file is small, you may do all of the above from within your program. Besides the same POKEs as above (you wouldn't need the filename of course), you will need the following line added to your program:

```
IF RC>1 THEN A=USR(1664,ADR(X$),RC)
```

where RC is the number of records stored in the string X\$. Substitute your names where applicable.

Programs 2,3,4, and 5 comprise a sort/merge utility that uses the same sort routine. This will give you the ability to handle much larger files and variable length records. With a 40 or 48K machine you will be able to sort files that are 60,000 bytes long. (If the record length is 60 characters, that will translate to 1,000 records.) This particular version divides the file into two manageable sub-files, sorts each, and then merges them. Be careful with

your disk space; the temporary file will need room also. If you have more than one drive, you can modify the program to split it three or more ways and sort even more records. For example, put the temporaries on drive 2 and the new file on drive 3. Who said micros can't handle larger files?

Your Options

The sort/merge utility is a stand-alone. Program 2 will load the machine language and display a title screen. Program 3 is a menu that will allow you to select either fixed or variable length record types and other parameters. If you select fixed length, Program 4 will be called; variable length will select Program 5.

Because of the chaining between these programs, Program 3 must be saved with a filename of "D:SORTXX". Programs 4 and 5 must likewise be saved with filenames of "D:SORT.FIX" and "D:SORT.VAR", respectively. Program 2 may be saved with any filename, but "D:SORTMERG" is suggested to avoid confusion.

Now that you know how to feed the sort its required parameters and call it, you must still get it into memory. Once again, you have several options. If you have the Assembler/Editor cartridge (or a similar assembler), the source appears in Program 6. Please feel free to modify it. If you're limited to BASIC, Program 7 will load the machine language when it is run. After doing either of these, you should go directly to DOS (DOS II only) and do a binary save (option K) with the following parameters:

```
D1:AUTORUN.SYS,0680,06FD
```

Saving the machine language as AUTORUN.SYS will enable the program to auto-boot when you power up with the disk (you *must* power up with that disk). Do *not* append an INIT or RUN address to the file unless you want the machine to lockup every time you turn it on. The stand-alone sort/merge utility will automatically load the machine language when RUN "D:SORTMERG" is executed by the Atari.

Program 1. Sort Program Load (Files)

```
10 REM CALLING PROGRAM MUST:
12 REM
14 REM *   POKE RECORD LENGTH INTO LOCATION 2
    05
15 REM *   POKE BEGINNING OF SORT KEY INTO LO
    C 203
16 REM *   POKE END OF SORT KEY INTO LOCATION
    204
```

```

17 REM * POKE TYPE (ASCENDING - 0 OR DESCEN
   DING - 1) INTO LOC 206
18 REM
19 REM THIS PROGRAM WILL LOAD FILE INTO MEMO
   RY AND CALL MACHINE
20 REM LANGUAGE ROUTINE. WHEN COMPLETED, YOU
   R PROGRAM MAY BE
21 REM RE-CALLED BY EQUATING P$ TO YOUR PROG
   RAM NAME.
22 REM
50 DIM X$(FRE(0)-600),R$(130),F$(15),P$(15),
   I$(1)
59 REM REPLACE X'S WITH YOUR FILE & PROGRAM
   NAMES
60 P$="XXXXXX":F$="XXXXXX"
99 REM GET RECORD LENGTH
100 RET=100:R=PEEK(205)
109 REM OPEN FILE AND INPUT RECORDS
110 ? " LOADING ";F$:TRAP 600:OPEN #2,4,0,F$
   :L=1
120 TRAP 140:INPUT #2,R$:TRAP 40000
130 X$(L,L+R-1)=R$:L=L+R:GOTO 120
140 CLOSE #2:L=L-1:N=L/R:?" RECORDS LOADED=
   ";N
149 REM CALL MACHINE LANGUAGE SORT ROUTINE
150 IF N>1 THEN ? " BEGIN SORT":A=USR(1664,A
   DR(X$),N)
160 RET=170:?" COMPLETED SAVING ";F$
169 REM ERASE OLD FILE AND SAVE NEW ONE
170 TRAP 600:XIO 36,#2,0,0,F$:OPEN #2,8,0,F$
180 FOR I=1 TO L STEP R:R$=X$(I,I+R-1):? #2;
   R$:NEXT I
190 CLOSE #2:XIO 35,#2,0,0,F$
199 REM RETURN TO YOUR PROGRAM ?
200 RET=200:TRAP 600:IF P$(3,4)<>"XX" THEN ?
   " LOADING ";P$:RUN P$
210 END
600 ? " ERROR - ";PEEK(195):CLOSE #2
610 ? " PRESS RETURN TO CONTINUE";:INPUT I$:
   GOTO RET

```

Program 2. Sort/Merge Loader

```

0 DIM M$(20):FOR I=1 TO 13:READ A:M$(I)=CHR$
  (A):NEXT I:DATA 72,198,208,165,208,141,10,
  212,141,24,208,104,64
1 GRAPHICS 21:POKE 752,1:POKE 82,1
2 POKE 708,52:POKE 709,8:POKE 710,148:POKE 7
  11,66:POKE 712,152:POKE 559,0

```

```

4 I=PEEK(560)+PEEK(561)*256:FOR J=1 TO 4:REA
  D A,B:POKE I+A,B:NEXT J
5 A=INT(ADR(M$)/256):POKE 513,A:POKE 512,ADR
  (M$)-A*256
6 FOR J=14 TO 30:POKE I+J,138:NEXT J:POKE 54
  286,192:POKE 559,34
8 DATA 3,70,6,6,7,6,8,6
10 POKE 87,2:POSITION 2,0:?"#6;"* Sort / Mer
GE *":?"#6;"{6 SPACES}UTILITY"
12 POKE 87,5
20 FOR N=1 TO 6:READ C,X1,Y1,X2,Y2,X3,Y3,X4,
  Y4
24 COLOR C:PLOT X1,Y1:DRAWTO X2,Y2:DRAWTO X3
  ,Y3:POSITION X4,Y4
26 POKE 765,C:XIO 18,#6,0,0,"S:":NEXT N
28 COLOR 2:FOR I=12 TO 27 STEP 3:PLOT 59,I:N
  EXT I
30 FOR Y=34 TO 38 STEP 2:COLOR 3:FOR X=15-Y+
  40 TO 62+Y-40 STEP 2:PLOT X,Y:NEXT X:COLO
  R 1:PLOT X+2,Y:NEXT Y
36 COLOR 4:PLOT 26,22:DRAWTO 26,14:DRAWTO 29
  ,14:PLOT 30,15:PLOT 31,16:PLOT 30,17:PLOT
  29,18
37 DRAWTO 27,18:DRAWTO 31,22:PLOT 34,14:DRAW
  TO 34,22:DRAWTO 39,22
38 PLOT 42,22:DRAWTO 42,14:DRAWTO 46,18:DRAW
  TO 50,14:DRAWTO 50,22
40 DATA 2,70,40,62,32,16,32,8,40,1,62,31,62,
  27,17,27,17,31,1,20,26,20,10,17,10,17,26
42 DATA 1,62,26,62,10,56,10,56,26,1,62,9,62,
  6,17,6,17,9,3,55,26,55,10,21,10,21,26
100 FOR I=0 TO 125:READ A:POKE 1664+I,A:NEXT I
102 POKE 54286,64:RUN "D:SortXX"
105 DATA 104,104,133,217,104,133,216,104,133
  ,209,104,133,208,169,0
110 DATA 133,218,133,207,162,1,165,216,133,2
  14,165,217,133,215,24
120 DATA 165,214,133,212,101,205,133,214,165
  ,215,133,213,105,0,133
130 DATA 215,164,203,165,206,240,10,177,214,
  209,212,144,44,240,12
140 DATA 176,19,177,214,209,212,144,13,240,2
  ,176,30,200,196,204
150 DATA 240,227,176,23,144,223,169,1,133,21
  8,164,205,136,177,214
160 DATA 72,177,212,145,214,104,145,212,192,
  0,208,241,232,224,0
170 DATA 208,2,230,207,228,208,208,172,165,2
  09,197,207,208,166,165
180 DATA 218,201,0,208,144,96

```

Program 3. Sort/Merge Menu (SAVE as "D:SORTXX")

```

0 REM SORT/MERGE MENU
10 POKE 82,1:GRAPHICS 0:?"{DOWN} SORT/MERGE
UTILITY":?"{DOWN}{TAB}"
20 DIM I$(1),T$(1):Q3=40000:?"{DOWN}FOR FILE
  TO BE SORTED, ENTER:"
30 ? "{DOWN}FIXED (F) or VARIABLE (V) LENGTH
  ";:INPUT I$
40 R=0:IF I$="V" THEN 70
50 IF I$<>"F" THEN 30
60 ? "RECORD LENGTH ";:TRAP 40:INPUT R:TRAP
  Q3:IF R<2 OR R>250 THEN 60
70 ? "SORT KEY (1st,2nd) ";:TRAP 70:INPUT SS
  ,SE:TRAP Q3
75 IF SS>=SE OR SS<0 OR SE>250 THEN 70
80 ? "ASCENDING - 0 OR DESCENDING - 1 ";:TR
  AP 80:INPUT T:TRAP Q3
85 IF T<0 OR T>1 THEN 80
90 POKE 205,R:POKE 203,SS:POKE 204,SE:POKE 2
  06,T
100 TRAP 120:IF I$="V" THEN RUN "D:SORT.VAR"
110 RUN "D:SORT.FIX"
120 ? "INSERT DISKETTE WITH SORT PROGRAM":?
  "PRESS RETURN ";:INPUT T$:GOTO 100

```

Program 4. Fixed Length Records (SAVE as "D:SORT.FIX")

```

0 REM SORT/MERGE - FIXED LENGTH RECORDS
20 R=PEEK(205):SS=PEEK(203)+1:SE=PEEK(204)+1
  :T=PEEK(206)
30 XL=FRE(0)-600:DIM X$(XL),F$(15),R$(R),T$(
  R),D$(7)
40 Q1=210:Q2=600:Q3=40000:D$="D1:TEMP"
50 ? "ENTER FILE NAME (Dn:name.ext)":INPUT F
  $
60 TRAP 50:DO=VAL(F$(2,2)):IF DO<1 OR DO>4 T
  HEN 50
80 ? "DRIVE NUMBER FOR SORTED FILE ";:TRAP 8
  0:INPUT DN
90 IF DN<1 OR DN>4 THEN 80
95 D$(2,2)=STR$(DO):?"INSERT ";F$;" IN DRIV
  E ";DO:IF DN<>DO THEN ? "AND BLANK DISK I
  N DRIVE ";DN
96 ? "PRESS RETURN ";:INPUT R$
100 ? "LOADING ";F$:TRAP Q2:OPEN #2,4,0,F$:M
  =0

```

```

120 L=1: ? "PASS 1 - ";:GOSUB 500:IF M=0 THEN
    160
140 ? "WRITING ";D$:OPEN #3,8,0,D$:GOSUB 560
150 ? "PASS 2 - ";:L=1:GOSUB 500
160 CLOSE #2:TRAP Q2:IF DO=DN THEN ? "DELETI
    NG ";F$:XIO 36,#3,0,0,F$
170 F$(2,2)=STR$(DN):OPEN #3,8,0,F$
180 ? "WRITING ";F$:IF M=0 THEN GOSUB 560:GO
    TO 400
200 TRAP Q2:OPEN #2,4,0,D$:J=1:A=1:B=1:AE=1:
    BE=1
210 IF A=1 THEN TRAP 330:INPUT #2,R$:TRAP Q3
220 IF B=1 THEN TRAP 340:T$=X$(J,J+R-1):J=J+
    R:TRAP Q3
230 IF AE=0 AND BE=0 THEN 390
240 IF AE=1 AND BE=0 THEN 300
245 IF AE=0 AND BE=1 THEN 310
250 IF T=1 THEN 280
260 IF R$(SS,SE)>T$(SS,SE) THEN 310
270 GOTO 300
280 IF R$(SS,SE)<T$(SS,SE) THEN 310
300 ? #3;R$:A=1:B=0:IF AE=0 THEN A=0:B=BE
302 GOTO Q1
310 ? #3;T$:A=0:B=1:IF BE=0 THEN B=0:A=AE
312 GOTO Q1
330 AE=0:GOTO 220
340 BE=0:GOTO 230
390 CLOSE #2: ? "DELETING ";D$:XIO 33,#2,0,0,
    D$
400 CLOSE #3:XIO 36,#3,0,0,F$
410 END
500 TRAP 530:INPUT #2,R$:TRAP Q3
510 X$(L)=R$:L=L+R:IF (L+R)<XL THEN 500
520 M=1
530 L=L-1:N=L/R: ? "RECORDS LOADED = ";N
540 IF N>1 THEN ? "BEGIN SORT ";:A=USR(1664
    ,ADR(X$),N)
550 ? "END SORT":RETURN
560 FOR I=1 TO L STEP R:R=X$(I,I+R-1): ? #3;
    R$:NEXT I:CLOSE #3:RETURN
600 ? "ERROR - ";PEEK(195):END

```

Program 5. Variable Length Records (SAVE as "D:SORT.VAR")

```

0 REM SORT/MERGE - VARIABLE LENGTH RECORDS
10 SS=PEEK(203)+1:SE=PEEK(204)+1:T=PEEK(206)
   :POKE 203,SS:POKE 204,SE
20 XL=FRE(0)-6000:DIM X$(XL),F$(15),R$(251),T
   $(251),D$(7)

```

```

30 Q1=210:Q2=600:Q3=40000:D$="D1:TEMP":T$(1)
   =" ":T$(251)=" ":T$(2)=T$(1)
40 ? "ENTER FILE NAME (Dn:name.ext)":INPUT F
   $
45 TRAP 40:DO=VAL(F$(2,2)):IF DO<1 OR DO>4 T
   HEN 40
50 ? "DRIVE NUMBER FOR SORTED FILE ";:TRAP 5
   0:INPUT DN
55 IF DN<1 OR DN>4 THEN 50
57 ? "INSERT ";F$;" IN DRIVE ";DO:IF DN<>DO
   THEN ? "AND BLANK DISK IN DRIVE ";DN
58 D$(2,2)=STR$(DO):? "PRESS RETURN ";:INPUT
   R$
60 ? "FINDING LONGEST RECORD LENGTH":TRAP Q2
   :OPEN #2,4,0,F$:R=0
70 TRAP 80:INPUT #2,R$:L=LEN(R$):IF L>R THEN
   R=L
75 GOTO 70
80 CLOSE #2: ? "LONGEST LENGTH IS ";R:IF R>25
   0 THEN ? "TOO LONG":END
100 POKE 205,R+1: ? "LOADING ";F$:TRAP Q2:OPE
   N #2,4,0,F$:M=0
120 L=1: ? "PASS 1 - ";:GOSUB 500:IF M=0 THEN
   160
140 ? "WRITING ";D$:OPEN #3,8,0,D$:GOSUB 560
150 ? "PASS 2 - ";:L=1:GOSUB 500
160 CLOSE #2:TRAP Q2:IF DO=DN THEN ? "DELETI
   NG ";F$:XIO 36,#3,0,0,F$
170 F$(2,2)=STR$(DN):OPEN #3,8,0,F$
180 ? "WRITING ";F$:IF M=0 THEN GOSUB 560:GO
   TO 400
200 TRAP Q2:OPEN #2,4,0,D$:J=1:A=1:B=1:AE=1:
   BE=1
210 IF A=1 THEN TRAP 330:INPUT #2,R$:TRAP Q3
220 IF B=1 THEN TRAP 340:RL=ASC(X$(J,J)):T$=
   X$(J+1,J+RL):J=J+R+1:TRAP Q3
230 IF AE=0 AND BE=0 THEN 390
240 IF AE=1 AND BE=0 THEN 300
245 IF AE=0 AND BE=1 THEN 310
250 IF T=1 THEN 280
260 IF R$(SS,SE)>T$(SS,SE) THEN 310
270 GOTO 300
280 IF R$(SS,SE)<T$(SS,SE) THEN 310
300 ? #3;R$:A=1:B=0:IF AE=0 THEN A=0:B=BE
302 GOTO Q1
310 ? #3;T$:A=0:B=1:IF BE=0 THEN B=0:A=AE
312 GOTO Q1
330 AE=0:GOTO 220
340 BE=0:GOTO 230

```

```

390 CLOSE #2: ? "DELETING "; D$: XIO 33, #2, 0, 0,
    D$
400 CLOSE #3: XIO 36, #3, 0, 0, F$
410 END
500 TRAP 530: INPUT #2, R$: TRAP Q3: RL = LEN(R$):
    IF RL < R THEN R$(RL+1) = T$
510 X$(L, L) = CHR$(RL): X$(L+1) = R$: L = L + R + 1: IF (
    L + R + 1) < XL THEN 500
520 M = 1
530 L = L - 1: N = L / (R + 1): ? "RECORDS LOADED = "; N
540 IF N > 1 THEN ? "BEGIN SORT "; A =USR(1664
    ,ADR(X$), N)
550 ? "END SORT": RETURN
560 FOR I = 1 TO L STEP R + 1: RL = ASC(X$(I, I)): R$
    = X$(I + 1, I + RL)
570 ? #3; R$: NEXT I: CLOSE #3: RETURN
600 ? "ERROR - "; PEEK(195): END

```

Program 6. Machine Language Bubble Sort

```

0100 .TITLE "MACHINE LANGUAGE BUBBLE SORT
"
0110 ;
0120 ; RLM MICRO SYSTEMS 01/20/82
0130 ;
0140 ; CALLED FROM BASIC WITH:
0150 ;
0160 ; A=USR(1664,ADR(X$),RC)
0170 ;
0180 ; NOTE: X$ IS THE STRING THAT CONTAINS
    THE FILE
0190 ; RC IS THE NUMBER OF RECORDS
0200 ;
0210 ; THE FOLLOWING ARE POKED BY BASIC PRO
    GRAM:
0220 ;
0230 ; SS - BEGINNING OF SORT KEY (DECI
    MAL- 203)
0240 ; SE - END OF SORT KEY (DECIMAL -
    204)
0250 ; RL - RECORD LENGTH (DECIMAL - 20
    5)
0260 ; TYPE - ASCENDING (0) OR DESCEND
    ING (1)
0270 ; (DECIMAL - 206)
0280 ;
0290 ; THE ROUTINE WILL LOOP THROUGH "FILE"
    SWAPPING UNSORTED
0300 ; ADJOINING MEMBERS UNTIL THE "SWAP" FL
    AG HAS NOT BEEN SET

```

```

0310 ; IN A GIVEN PASS. THE ZERO PAGE ADDRESS
      SES "FST" AND "SEC"
0320 ; POINT AT THE INDIVIDUAL MEMBERS BEING
      COMPARED. THE Y
0330 ; REGISTER IS USED AS AN INDEX POINTER
      FOR TESTING OR
0340 ; MOVING BYTES BETWEEN THE TWO RECORDS.
0350 ;
0360      *=      $0680      START ON PAGE 6
0370 FST      =      $D4      MEMBER n ADDRESS (LS
      B,MSB)
0380 SEC      =      $D6      MEMBER (n+1) ADDRESS
      (LSB,MSB)
0390 BASE     =      $D8      BASE ADDRESS OF LIST
      (LSB,MSB)
0400 SS       =      $CB      FIRST POSITION OF SO
      RT KEY
0410 SE       =      $CC      LAST POSITION OF SOR
      T KEY
0420 RL       =      $CD      ELEMENT LENGTH
0430 SWAP     =      $DA      SWAP SWITCH
0440 RC       =      $D0      NUMBER OF ELEMENTS (
      LSB,MSB)
0450 CNTH     =      $CF      RECORD COUNTER (MSB,
      X REG IS LSB)
0460 TYPE     =      $CE      SORT TYPE, 0-ASC 1-
      DES
0470 ;
0480 ;
0490      PLA      POP # OF ARGUMENTS F
      ROM STACK
0500      PLA
0510      STA      BASE+1      SET BASE ADDRESS
0520      PLA
0530      STA      BASE
0540      PLA
0550      STA      RC+1      SET ELEMENT COUNT
0560      PLA
0570      STA      RC
0580 ;
0590 ;
0600 BEGIN LDA # $00      START EACH PASS THRO
      UGH FILE
0610      STA      SWAP      SET SWAP TO 0
0620      STA      CNTH      SET HIGH COUNT TO 0
0630      LDX      # $01      SET X REGISTER TO 1
      (LOW COUNT)
0640      LDA      BASE      SET POINTER (n) TO B
      ASE

```



```

0650      STA  SEC
0660      LDA  BASE+1
0670      STA  SEC+1
0680      ;
0690  CONT  CLC
0700      LDA  SEC          RESET POINTERS-
0710      STA  FST          (n) to (n+1)
0720      ADC  RL
0730      STA  SEC          (n+1) to (n+2)
0740      LDA  SEC+1
0750      STA  FST+1
0760      ADC  #$00
0770      STA  SEC+1
0780      LDY  SS          ASCII STRING COMPARI
SON
0790      ;
0800  COMP  LDA  TYPE      ASCENDING OR DESCEND
ING?
0810      BEQ  ASC          SORT IS ASCENDING
0820      LDA  (SEC),Y      TYPE = DESCENDING
0830      CMP  (FST),Y      COMPARE ADJOINING ME
MBERS
0840      BCC  BACK        (n) > (n+1)
0850      BEQ  INCR        (n) = (n+1) TRY AGAIN
0860      BCS  FLIP        (n) < (n+1)
0870      ;
0880  ASC   LDA  (SEC),Y    TYPE = ASCENDING
0890      CMP  (FST),Y    COMPARE ADJOINING ME
MBERS
0900      BCC  FLIP        (n) > (n+1)
0910      BEQ  INCR        (n) = (n+1) TRY AGAIN
0920      BCS  BACK        (n) < (n+1)
0930      ;
0940  INCR  INY            ADD 1 TO POINTER
0950      CPY  SE          END OF SORT KEY?
0960      BEQ  COMP        NO
0970      BCS  BACK        YES, NEXT ELEMENT
0980      BCC  COMP        NO
0990      ;
1000  FLIP  LDA  #$01      SWAP ELEMENTS (n), (n
+1)
1010      STA  SWAP        SET SWAP SWITCH ON
1020      LDY  RL          LOAD LENGTH
1030      ;
1040  MOVE  DEY            SET DISPLACEMENT
1050      LDA  (SEC),Y      EXCHANGE BYTES
1060      PHA
1070      LDA  (FST),Y
1080      STA  (SEC),Y

```

```

1090      PLA
1100      STA (FST),Y
1110      CPY #000      MORE BYTES TO SWAP?
1120      BNE MOVE      YES
1130 ;
1140 BACK INX          INCREMENT RECORD COU
      NTER
1150      CPX #000      CHECK FOR >255
1160      BNE TEST
1170      INC CNTH      ADD 1 TO HIGH COUNT
1180 ;
1190 TEST CPX RC        END OF FILE?
1200      BNE CONT      NO
1210      LDA RC+1      CHECK HIGH EOF
1220      CMP CNTH
1230      BNE CONT      NOT END OF FILE
1240      LDA SWAP      TEST FOR END OF SORT
1250      CMP #000      ANY SWAPS?
1260      BNE BEGIN     YES, START OVER
1270      RTS           NO, RETURN TO CALLIN
      G PROGRAM
1280      .END

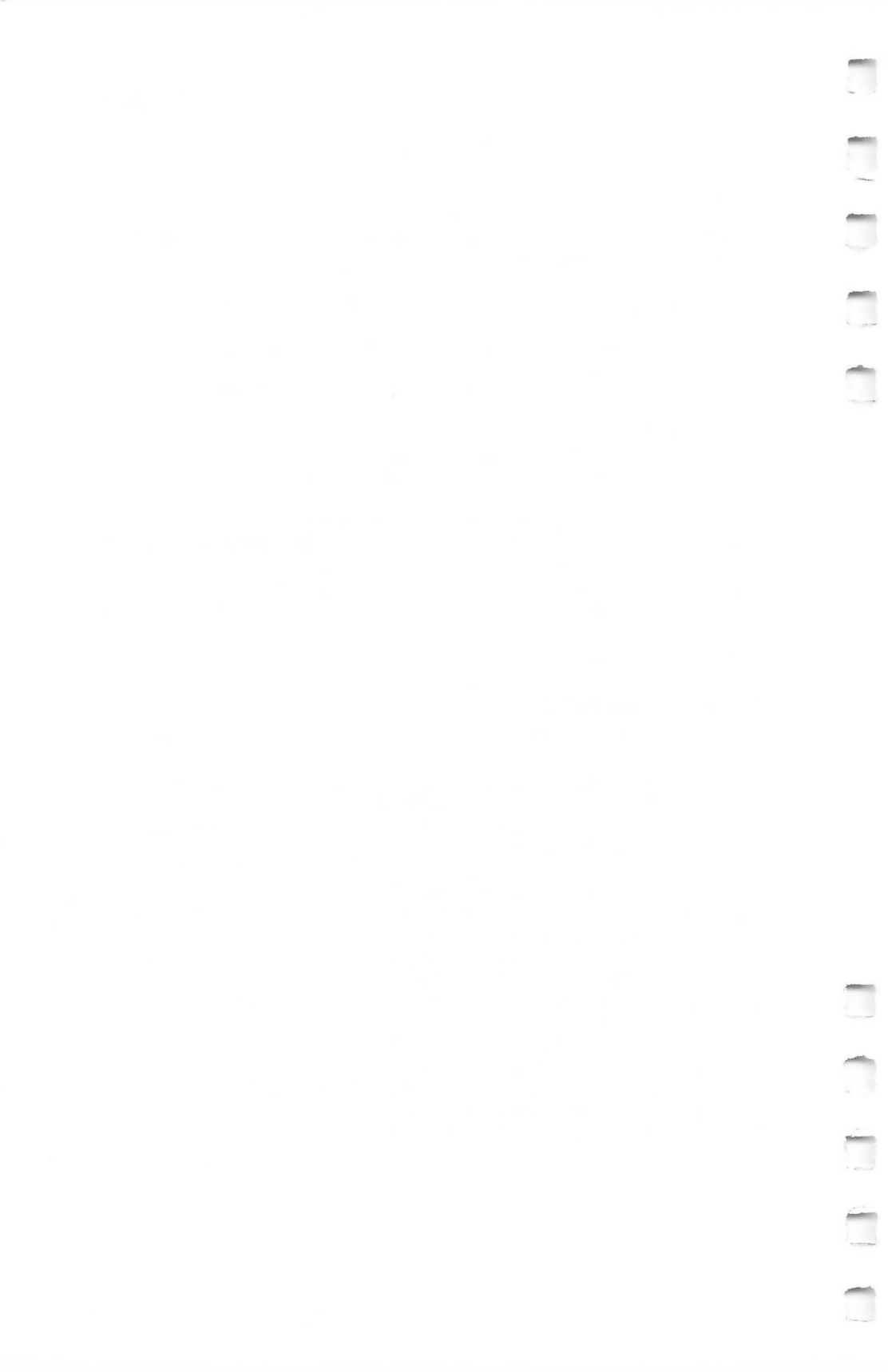
```

Program 7. Sort Load

```

98 FOR I=0 TO 125:READ A:POKE 1664+I,A:NEXT
  I
100 DATA 104,104,133,217,104,133,216,104,133
      ,209,104,133,208,169,0
110 DATA 133,218,133,207,162,1,165,216,133,2
      14,165,217,133,215,24
120 DATA 165,214,133,212,101,205,133,214,165
      ,215,133,213,105,0,133
130 DATA 215,164,203,165,206,240,10,177,214,
      209,212,144,44,240,12
140 DATA 176,19,177,214,209,212,144,13,240,2
      ,176,30,200,196,204
150 DATA 240,227,176,23,144,223,169,1,133,21
      8,164,205,136,177,214
160 DATA 72,177,212,145,214,104,145,212,192,
      0,208,241,232,224,0
170 DATA 208,2,230,207,228,208,208,172,165,2
      09,197,207,208,166,165
180 DATA 218,201,0,208,144,96

```



A Appendix

A

A Complete Guide to the Atari Character Set

Orson Scott Card

Atari characters can be used to do many things besides speak English to the user. Nearly infinite strings can hold fully relocatable machine language programs in character form, the most economical way of storing machine language in BASIC programs. Characters can be POKEd directly into screen memory. Programs can read the keyboard directly, by-passing the Atari's keyboard handling routines, so that you can effectively redefine almost every key and key combination. And editing functions, like CURSOR LEFT, DELETE, CLEAR, TAB, and even TAB SET and TAB CLEAR, can all be executed during a program simply by PRINTing them, either as part of a string or as a CHR\$(n) function.

The trouble is, to do all these things requires using several different codes. And the different codes have all been kept in different lists—often in different books—and as often as not you've had to translate hexadecimal to decimal or multiply by 8 in order to get the value you wanted.

Until now. Here is a complete listing of the Atari character set, in ATASCII order, with every bit of information we could think of a use for. For each of 128 characters, you will find:

- The pattern of on-off bits that produces the character on the screen, including the value of each byte in the pattern.
- The ATASCII values in decimal and hexadecimal for regular and inverse characters.
- The internal code values in decimal and hexadecimal for regular and inverse characters.
- The keyboard code values in decimal and hexadecimal, including the value of the key combination and the value of the unshifted key alone.
- The machine language instruction represented by the regular and/or inverse character's ATASCII value.

- The offset of the character's 8-byte pattern within character set memory.
- The key combination required to PRINT the character (or execute its screen editing function).
- The effect of PRINTing screen editing characters.

How to Use the Table

Each entry begins with a printout showing the pattern of on-off bits in the character pattern. Beside each row is the value, in decimal, of the byte that produces that row's on-off pattern. On bits are 1, off bits are 0. The operating system creates inverse characters from the same patterns, except that 0 is interpreted as on and 1 is interpreted as off.

ATASCII VALUE

The first line gives the ATASCII code in decimal and hexadecimal (\$) and the value of the inverse character. If the character is also an editing command, the effect of PRINTing the character is given in the third column of the first line.

Machine Language

The second line gives the 6502 machine language instruction represented by the ATASCII value of the character, followed by the instruction represented by the ATASCII value of the inverse character. If the inverse character is also an editing command, the effect of PRINTing the inverse character is given in the third column of the second line.

The following conventions are used with the machine language mnemonics:

= immediate addressing
 z = absolute zero page addressing
 abs = absolute 2-byte addressing
 (ind) = indirect addressing
 ,X or ,Y = indexed addressing
 A = accumulator

Remember that the machine language mnemonic represents the ATASCII value of the character, not the ICODE (internal code) value. This information is provided so you can decode machine language routines contained in strings, like:

C = USR(ADR("string"))

Also, keep in mind that after almost every instruction comes a 1- or 2-byte argument. Any instruction that uses absolute

addressing will be followed by a 2-byte argument; instructions that use indirect, zero page, and immediate addressing, as well as branch instructions, will use 1-byte arguments; and instructions with implied addressing (DEY, INX, RTS, NOP, BRK, etc.) will have no argument following them.

ICODE Values

The third line gives the ICODE (internal code) value of the character. This is the number that must be POKEd into screen memory to display the character on the screen; the number also represents the order of the character within character set memory. The ICODE value is given in decimal and hexadecimal, followed by the ICODE value of the inverse character in decimal and hexadecimal. Last comes the offset of the character in the character set—the number of bytes to count into character set memory to find the top line of that character's pattern.

(Occasionally the keyboard code is also called an internal code, but for clarity we will use ICODE only for the number representing the character's order in character set memory, which is also the number POKEd into screen memory.)

KEYCODE Values

The fourth line gives the KEYCODE (keyboard code) value of the character—the number that is stored in location 764 when you press the key combination that produces that character. The number is given in decimal and hexadecimal, followed by the decimal and hexadecimal *unshifted* KEYCODE—the code for the individual key, regardless of whether SHIFT or CONTROL are pressed. Last comes the key combination required to produce the character. If the character is also an editing command, (ESC) will come first to remind you to PRINT or type the ESC character first or PRINTing the character will execute its editing function.

Indexes

To help you use this table, it is followed by several indexes:

ICODE index. Look up characters by their internal code number.

Machine language index. Look up characters by the machine language mnemonic (in alphabetical order).

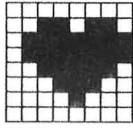
KEYCODE index. Look up characters by their keyboard code number.

Screen Editing Characters

The screen editing characters are paired, with the second character in each pair PRINTed as the inverse of the first character. To be PRINTed—PRINT CHR\$(*nnn*)—each character must be preceded by the ESC character—PRINT CHR\$(27);CHR\$(*nnn*). The only exception is CHR\$(155), the RETURN character. If you could PRINT it, it would be the inverse ESCAPE character, which is what appears when you POKE the ICODE equivalent, 219, into screen memory. However, used with PRINT, CHR\$(155) will always cause the Operating System to execute a carriage return and line feed. There is no way to defeat this without altering the OS.

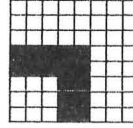
Normal	Inverse
27	155
ESC	RETURN
28	156
CURSOR UP	DELETE LINE
29	157
CURSOR DOWN	INSERT LINE
30	158
CURSOR LEFT	TAB CLEAR
31	159
CURSOR RIGHT	TAB SET
125	253
CLEAR	BUZZER [CONTROL-2]
126	254
DELETE BACK	DELETE AHEAD
127	255
TAB	INSERT CHARACTER

Table 1. Atari Character Set



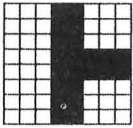
0
1
2
3
4
5
6
7

ATASCII 0 \$00 inv 128 \$80
ML BRK inv —
ICODE 64 \$40 inv 192 \$C0 offset 512
KEYCODE 160 \$A0 uns 32 \$20 CONTROL-



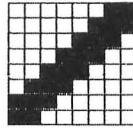
0
1
2
3
4
5
6
7

ATASCII 5 \$05 inv 133 \$85
ML ORA z inv STA z
ICODE 69 \$45 inv 197 \$C5 offset 552
KEYCODE 170 \$AA uns 42 \$2A CONTROL-E



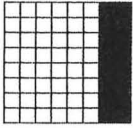
2
3
4
5
6
7

ATASCII 1 \$01 inv 129 \$80
ML ORA (ind,X) inv STA (ind,X)
ICODE 65 \$41 inv 193 \$C1 offset 520
KEYCODE 191 \$BF uns 63 \$BF CONTROL-A



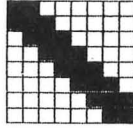
3
4
5
6
7

ATASCII 6 \$06 inv 134 \$86
ML ASL z inv STX z
ICODE 70 \$46 inv 198 \$C6 offset 560
KEYCODE 184 \$B8 uns 56 \$38 CONTROL-F



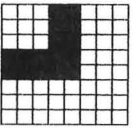
0
1
2
3
4
5
6
7

ATASCII 2 \$02 inv 130 \$82
ML — inv —
ICODE 66 \$42 inv 194 \$C2 offset 520
KEYCODE 149 \$95 uns 21 \$15 CONTROL-B



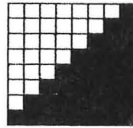
1
2
3
4
5
6
7

ATASCII 7 \$07 inv 135 \$87
ML — inv —
ICODE 71 \$47 inv 199 \$C7 offset 568
KEYCODE 189 \$BD uns 61 \$3D CONTROL-G



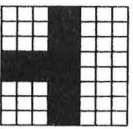
2
3
4
5
6
7

ATASCII 3 \$03 inv 131 \$83
ML — inv —
ICODE 67 \$43 inv 195 \$C3 offset 536
KEYCODE 146 \$92 uns 18 \$12 CONTROL-C



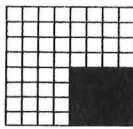
1
2
3
4
5
6
7

ATASCII 8 \$08 inv 136 \$88
ML PHP inv DEY
ICODE 72 \$48 inv 200 \$C8 offset 576
KEYCODE 185 \$B9 uns 57 \$39 CONTROL-H



2
3
4
5
6
7

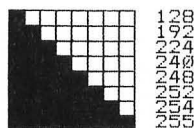
ATASCII 4 \$04 inv 132 \$84
ML — inv STY z
ICODE 68 \$44 inv 196 \$C4 offset 544
KEYCODE 186 \$BA uns 58 \$3A CONTROL-D



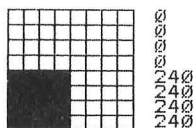
0
1
2
3
4
5
6
7

ATASCII 9 \$09 inv 137 \$89
ML ORA # inv —
ICODE 73 \$49 inv 201 \$C9 offset 584
KEYCODE 141 \$8D uns 13 \$0D CONTROL-I

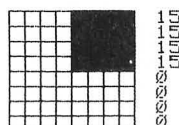
Table 1. Atari Character Set (continued)



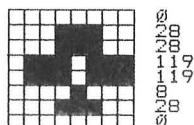
ATASCII 10 \$0A inv 138 \$8A
ML ASL A inv TXA
ICODE 74 \$4A inv 202 \$CA offset 592
KEYCODE 129 \$81 uns 1 \$01 CONTROL-J



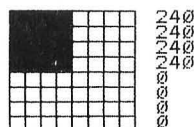
ATASCII 15 \$0F inv 143 \$8F
ML — inv —
ICODE 79 \$4F inv 207 \$CF offset 632
KEYCODE 136 \$88 uns 8 \$08 CONTROL-O



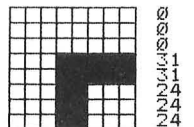
ATASCII 11 \$0B inv 139 \$8B
ML — inv —
ICODE 75 \$4B inv 203 \$CB offset 600
KEYCODE 133 \$85 uns 5 \$05 CONTROL-K



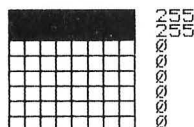
ATASCII 16 \$10 inv 144 \$90
ML BPL inv BCC
ICODE 80 \$50 inv 208 \$D0 offset 640
KEYCODE 138 \$8A uns 10 \$0A CONTROL-P



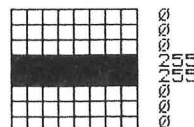
ATASCII 12 \$0C inv 140 \$8C
ML — inv STY abs
ICODE 76 \$4C inv 204 \$CC offset 608
KEYCODE 128 \$80 uns 0 \$00 CONTROL-L



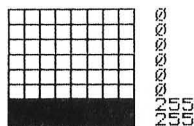
ATASCII 17 \$11 inv 145 \$91
ML ORA (ind),Y inv STA (ind),Y
ICODE 81 \$51 inv 209 \$D1 offset 648
KEYCODE 175 \$AF uns 47 \$2F CONTROL-Q



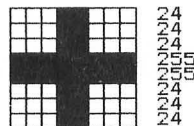
ATASCII 13 \$0D inv 141 \$8D
ML ORA abs inv STA abs
ICODE 77 \$4D inv 205 \$CD offset 616
KEYCODE 165 \$A5 uns 37 \$25 CONTROL-M



ATASCII 18 \$12 inv 146 \$92
ML — inv —
ICODE 82 \$52 inv 210 \$D2 offset 656
KEYCODE 168 \$A8 uns 40 \$28 CONTROL-R



ATASCII 14 \$0E inv 142 \$8E
ML ASL abs inv STX abs
ICODE 78 \$4E inv 206 \$CE offset 624
KEYCODE 163 \$A3 uns 35 \$23 CONTROL-N



ATASCII 19 \$13 inv 147 \$93
ML — inv —
ICODE 83 \$53 inv 211 \$D3 offset 664
KEYCODE 190 \$BE uns 62 \$3E CONTROL-S

Table 1. Atari Character Set (continued)

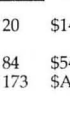
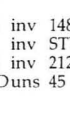
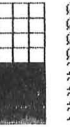

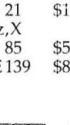
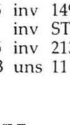
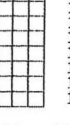
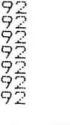
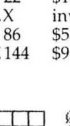
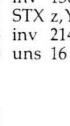
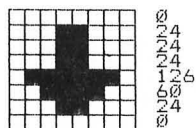
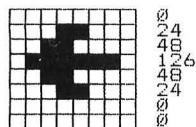
		ATASCII 20 ML — ICODE 84 KEYCODE 173	\$14 inv inv \$54 inv \$ADuns	148 STY 212 45	\$94 z,X \$D4 offset 672 \$2D CONTROL-T
		ATASCII 21 ML ORA z,X ICODE 85 KEYCODE 139	\$15 inv inv \$55 inv \$8B uns	149 STA 213 11	\$95 z,X \$D5 offset 680 \$0B CONTROL-U
		ATASCII 22 ML ASL z,X ICODE 86 KEYCODE 144	\$16 inv inv \$56 inv \$90 uns	150 STX z,Y 214 16	\$96 z,Y \$D6 offset 688 \$10 CONTROL-V
		ATASCII 23 ML — ICODE 87 KEYCODE 174	\$17 inv inv \$57 inv \$AE uns	151 — 215 46	\$97 — \$D7 offset 696 \$2E CONTROL-W
		ATASCII 24 ML CLC ICODE 88 KEYCODE 150	\$18 inv inv \$58 inv \$96 uns	152 TYA 216 22	\$98 — \$D8 offset 704 \$16 CONTROL-X

Table 1. Atari Character Set (continued)



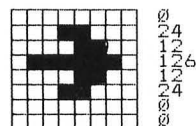
0
24
24
24
126
60
24
0

ATASCII 29 \$1D inv 157 \$9D CURSOR
DOWN
ML ORA abs,X inv STA abs,X INSERT LINE
ICODE 93 \$5D inv 221 \$DD offset 744
KEYCODE143 \$8F uns 15 \$0F {ESC}
CONTROL- =



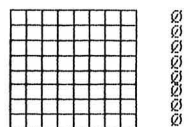
0
24
48
126
148
24
0

ATASCII 30 \$1E inv 158 \$9E CURSOR LEFT
CLEAR TAB
ML ASL abs,X inv —
ICODE 94 \$5E inv 222 \$DE offset 752
KEYCODE134 \$86 uns 6 \$06 {ESC}
CONTROL- +



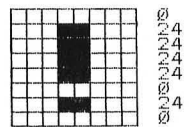
0
24
126
126
126
24
0

ATASCII 31 \$1F inv 159 \$9F CURSOR
RIGHT
ML — inv —
SET TAB
ICODE 95 \$5F inv 223 \$DF offset 769
KEYCODE135 \$87 uns 7 \$07 {ESC}
CONTROL-*



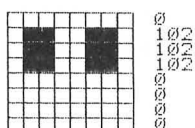
0
0
0
0
0
0
0

ATASCII 32 \$20 inv 160 \$A0
ML JSR abs inv LDY #
ICODE 0 \$00 inv 128 \$80 offset 0
KEYCODE33 \$21 uns 33 \$21 SPACE BAR



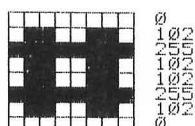
0
24
24
24
24
24
0

ATASCII 33 \$21 inv 161 \$A1
ML AND (ind,X) inv LDA (ind,X)
ICODE 1 \$01 inv 129 \$81 offset 8
KEYCODE95 \$5F uns 31 \$1F SHIFT-1



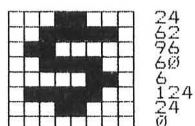
0
102
102
102
102
0
0
0
0

ATASCII 34 \$22 inv 162 \$A2
ML — inv LDX #
ICODE 2 \$02 inv 130 \$82 offset 16
KEYCODE94 \$5E uns 30 \$1E SHIFT-2



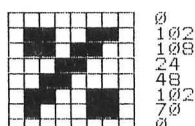
0
102
255
102
102
255
102
0

ATASCII 35 \$23 inv 163 \$A3
ML — inv —
ICODE 3 \$03 inv 131 \$83 offset 24
KEYCODE90 \$5A uns 26 \$1A SHIFT-3



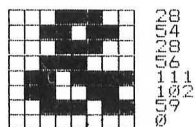
24
62
96
60
6
124
24
0

ATASCII 36 \$24 inv 164 \$A4
ML BIT z inv LDY z
ICODE 4 \$04 inv 133 \$84 offset 32
KEYCODE88 \$58 uns 24 \$18 SHIFT-4



0
102
108
24
48
102
70
0

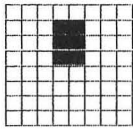
ATASCII 37 \$25 inv 165 \$A5
ML AND z inv LDA z
ICODE 5 \$05 inv 133 \$85 offset 40
KEYCODE93 \$5D uns 29 \$1D SHIFT-5



28
54
28
56
111
102
59
0

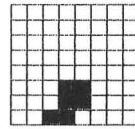
ATASCII 38 \$26 inv 166 \$A6
ML ROL z inv LDX z
ICODE 6 \$06 inv 134 \$86 offset 48
KEYCODE91 \$5B uns 27 \$1B SHIFT-6

Table 1. Atari Character Set (continued)



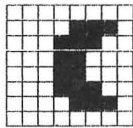
0
4
4
4
4
0
0
0
0

ATASCII 39 \$27 inv 167 \$A7
ML — inv —
ICODE 7 \$07 inv 135 \$87 offset 56
KEYCODE 115 \$73 uns 51 \$33 SHIFT-7



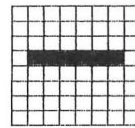
0
0
0
0
4
4
4
4
0

ATASCII 44 \$2C inv 172 \$AC
ML BIT abs inv LDY abs
ICODE 12 \$0C inv 140 \$8C offset 96
KEYCODE 32 \$20 uns 32 \$20 , [comma]



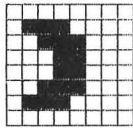
0
14
28
24
24
28
14
0

ATASCII 40 \$28 inv 168 \$A8
ML PLP inv TAY
ICODE 8 \$08 inv 136 \$88 offset 64
KEYCODE 112 \$70 uns 48 \$30 SHIFT-9



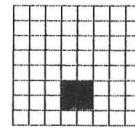
0
0
126
0
0
0
0
0

ATASCII 45 \$2D inv 173 \$AD
ML AND abs inv LDA abs
ICODE 13 \$0D inv 141 \$8D offset 104
KEYCODE 14 \$0E uns 14 \$0E - [hyphen]



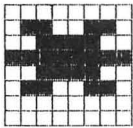
0
12
56
24
24
56
12
0

ATASCII 41 \$29 inv 169 \$A9
ML AND # inv LDA #
ICODE 9 \$09 inv 137 \$89 offset 72
KEYCODE 114 \$72 uns 50 \$32 SHIFT-0



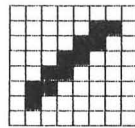
0
0
0
0
0
4
4
0

ATASCII 46 \$2E inv 174 \$AE
ML ROL abs inv LDX abs
ICODE 14 \$0E inv 142 \$8E offset 112
KEYCODE 34 \$22 uns 34 \$22



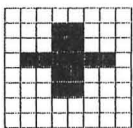
0
102
505
255
505
102
0

ATASCII 42 \$2A inv 170 \$AA
ML ROL A inv TAX
ICODE 10 \$0A inv 138 \$8A offset 80
KEYCODE 7 \$07 uns 7 \$07 *



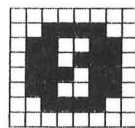
0
5
12
24
48
96
64
0

ATASCII 47 \$2F inv 175 \$AF
ML — inv —
ICODE 15 \$0F inv 143 \$8F offset 120
KEYCODE 38 \$26 uns 38 \$26 /



0
4
24
24
126
24
4
0

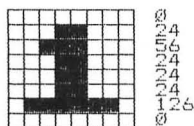
ATASCII 43 \$2B inv 171 \$AB
ML — inv —
ICODE 11 \$0B inv 139 \$8B offset 88
KEYCODE 6 \$06 uns 6 \$06 +



0
602
1002
11002
11002
1002
602
0

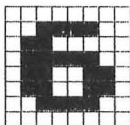
ATASCII 48 \$30 inv 176 \$B0
ML BMI inv BCS
ICODE 16 \$10 inv 144 \$90 offset 128
KEYCODE 50 \$32 uns 50 \$32 0

Table 1. Atari Character Set (continued)



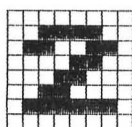
0 4
2 6
2 4
2 4
2 4
1 2 6
0

ATASCII 49 \$31 inv 177 \$B1
ML AND (ind),Y inv LDA (ind),Y
ICODE 17 \$11 inv 145 \$91 offset 136
KEYCODE31 \$1F uns 31 \$1F 1



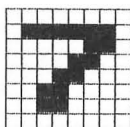
0 6
6 6
1 2 4
1 0 2 2
1 0 2
6 0
0

ATASCII 54 \$36 inv 182 \$B6
ML ROL z,X inv LDX z,Y
ICODE 22 \$16 inv 150 \$96 offset 176
KEYCODE27 \$1B uns 27 \$1B 6



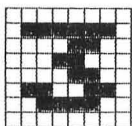
0 6
6 0 2
1 0 2
1 2 4
4 8
1 2 6
0

ATASCII 50 \$32 inv 178 \$B2
ML — inv —
ICODE 18 \$12 inv 146 \$92 offset 144
KEYCODE30 \$1E uns 30 \$1E 2



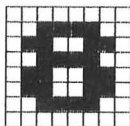
0 1 2 6
6 2
1 2
1 2 4
4 8
4 8
0

ATASCII 55 \$37 inv 183 \$B7
ML — inv —
ICODE 23 \$17 inv 151 \$97 offset 184
KEYCODE51 \$33 uns 51 \$33 7



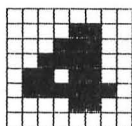
0 1 2 6
1 2 6
1 2 4
2 2 4
1 0 2
1 0 2
6 0
0

ATASCII 51 \$33 inv 179 \$B3
ML — inv —
ICODE 19 \$13 inv 147 \$93 offset 152
KEYCODE26 \$1A uns 26 \$1A 3



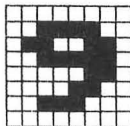
0 6
6 0 2
1 0 2
6 0
1 0 2
1 0 2
6 0
0

ATASCII 56 \$38 inv 184 \$B8
ML SEC inv CLV
ICODE 24 \$18 inv 152 \$98 offset 192
KEYCODE53 \$35 uns 53 \$35 8



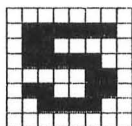
0 1 2
1 2 6
2 6
1 0 8
1 0 8
1 2 6
1 2 6
0

ATASCII 52 \$34 inv 180 \$B4
ML — inv —
ICODE 20 \$14 inv 148 \$94 offset 160
KEYCODE24 \$18 uns 24 \$18 4



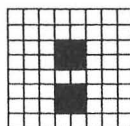
0 6
6 0 2
1 0 2
6 2
6 1 2
5 6
0

ATASCII 57 \$39 inv 185 \$B9
ML AND abs,Y inv LDA abs,Y
ICODE 25 \$19 inv 153 \$99 offset 200
KEYCODE48 \$30 uns 48 \$30 9



0 1 2 6
9 6
1 2 4
6
1 0 2
6 0
0

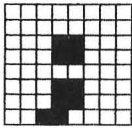
ATASCII 53 \$35 inv 181 \$B5
ML AND z,X inv LDA z,X
ICODE 21 \$15 inv 149 \$95 offset 168
KEYCODE29 \$1D uns 29 \$1D 5



0 2 4
2 4
0 2 4
2 4
0 2 4
0

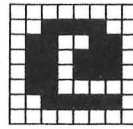
ATASCII 58 \$3A inv 186 \$BA
ML — inv TSX
ICODE 26 \$1A inv 154 \$9A offset 208
KEYCODE66 \$42 uns 2 \$02 SHIFT-;

Table 1. Atari Character Set (continued)



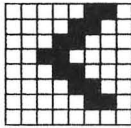
0
0
24
24
0
24
24
48

ATASCII 59 \$3B inv 187 \$BB
ML — inv —
ICODE 27 \$1B inv 155 \$9B offset 216
KEYCODE2 \$02 uns 2 \$02 ;



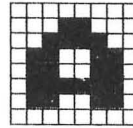
0
60
102
110
110
110
96
62
0

ATASCII 64 \$40 inv 192 \$C0
ML RTI inv CPY #
ICODE 32 \$20 inv 160 \$A0 offset 256
KEYCODE117 \$75 uns 53 \$35 SHIFT-8



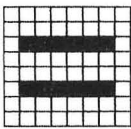
6
12
24
48
24
12
6
0

ATASCII 60 \$3C inv 188 \$BC
ML — inv LDY abs,X
ICODE 28 \$1C inv 156 \$9C offset 224
KEYCODE54 \$36 uns 54 \$36 <



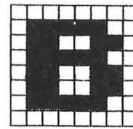
0
24
60
102
102
126
102
0

ATASCII 65 \$41 inv 193 \$C1
ML EOR (ind,X) inv CMP (ind,X)
ICODE 33 \$21 inv 161 \$A1 offset 264
KEYCODE127 \$7F uns 63 \$3F SHIFT-A



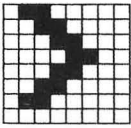
0
0
126
0
126
0
0

ATASCII 61 \$3D inv 189 \$BD
ML AND abs,X inv LDA abs,X
ICODE 29 \$1D inv 157 \$9D offset 232
KEYCODE15 \$0F uns 15 \$0F =



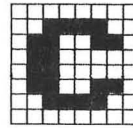
0
124
102
124
102
102
124
0

ATASCII 66 \$42 inv 194 \$C2
ML — inv —
ICODE 34 \$22 inv 162 \$A2 offset 272
KEYCODE85 \$55 uns 21 \$15 SHIFT-B



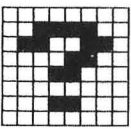
96
48
24
12
24
48
96
0

ATASCII 62 \$3E inv 190 \$BE
ML ROL abs,X inv LDX abs,Y
ICODE 30 \$1E inv 158 \$9E offset 240
KEYCODE55 \$37 uns 55 \$37 >



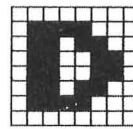
0
60
102
96
96
102
60
0

ATASCII 67 \$43 inv 195 \$C3
ML — inv —
ICODE 35 \$23 inv 163 \$A3 offset 280
KEYCODE82 \$52 uns 18 \$12 SHIFT-C



0
60
102
124
24
24
0

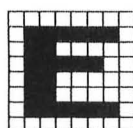
ATASCII 63 \$3F inv 191 \$BF
ML — inv —
ICODE 31 \$1F inv 159 \$9F offset 248
KEYCODE102 \$66 uns 38 \$26 SHIFT-/



0
120
108
102
102
108
120
0

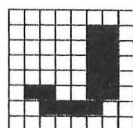
ATASCII 68 \$44 inv 196 \$C4
ML — inv CPY z
ICODE 36 \$24 inv 164 \$A4 offset 288
KEYCODE122 \$7A uns 58 \$3A SHIFT-D

Table 1. Atari Character Set (continued)



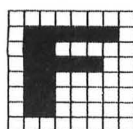
0
1 2 6
9 6
1 2 4
9 6
9 6
1 2 6
0

ATASCII 69 \$45 inv 197 \$C5
ML EOR z inv CMP z
ICODE 37 \$25 inv 165 \$A5 offset 296
KEYCODE 106 \$6A uns 42 \$2A SHIFT-E



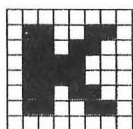
0
6
6
6
6
1 0 2
6 0
0

ATASCII 74 \$4A inv 202 \$CA
ML LSR A inv DEX
ICODE 42 \$2A inv 170 \$AA offset 336
KEYCODE 65 \$41 uns 1 \$01 SHIFT-J



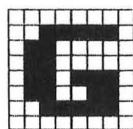
0
1 2 6
9 6
1 2 4
9 6
9 6
9 6
0

ATASCII 70 \$46 inv 198 \$C6
ML LSR z inv DEC z
ICODE 38 \$26 inv 166 \$A6 offset 304
KEYCODE 120 \$78 uns 56 \$38 SHIFT-F



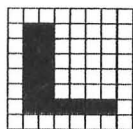
0
1 0 2
1 0 8
1 2 0
1 2 0
1 0 8
1 0 2
0

ATASCII 75 \$4B inv 203 \$CB
ML — inv —
ICODE 43 \$2B inv 171 \$AB offset 344
KEYCODE 69 \$45 uns 5 \$05 SHIFT-K



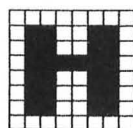
0
6 2
9 6
9 6
9 6
1 1 0
1 0 2
6 2
0

ATASCII 71 \$47 inv 199 \$C7
ML — inv —
ICODE 39 \$27 inv 167 \$A7 offset 312
KEYCODE 125 \$7D uns 61 \$3D SHIFT-G



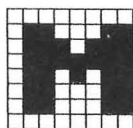
0
9 6
9 6
9 6
9 6
9 6
1 2 6
0

ATASCII 76 \$4C inv 204 \$CC
ML JMP abs inv CPY abs
ICODE 44 \$2C inv 172 \$AC offset 352
KEYCODE 64 \$40 uns 0 \$00 SHIFT-L



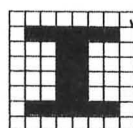
0
1 0 2
1 0 2
1 2 6
1 0 2
1 0 2
1 0 2
0

ATASCII 72 \$48 inv 200 \$C8
ML PHA inv INY
ICODE 40 \$28 inv 168 \$A8 offset 320
KEYCODE 121 \$79 uns 57 \$39 SHIFT-H



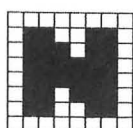
0
9 9
1 1 9
1 2 7
1 0 7
9 9
9 9
0

ATASCII 77 \$4D inv 205 \$CD
ML EOR abs inv CMP abs
ICODE 45 \$2D inv 173 \$AD offset 360
KEYCODE 101 \$65 uns 37 \$25 SHIFT-M



0
1 2 6
2 4
2 4
2 4
2 4
1 2 6
0

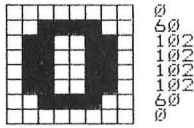
ATASCII 73 \$49 inv 201 \$C9
ML EOR # inv CMP #
ICODE 41 \$29 inv 169 \$A9 offset 328
KEYCODE 77 \$4D uns 13 \$0D SHIFT-I



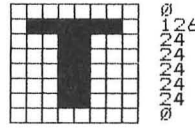
0
1 0 2
1 1 8
1 2 6
1 2 6
1 1 8
1 0 2
0

ATASCII 78 \$4E inv 206 \$CE
ML LSR abs inv DEC z,X
ICODE 46 \$2E inv 174 \$AE offset 368
KEYCODE 99 \$63 uns 35 \$23 SHIFT-N

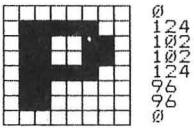
Table 1. Atari Character Set (continued)



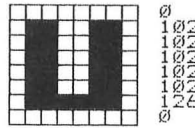
ATASCII 79 \$4F inv 207 \$CF
ML — inv —
ICODE 47 \$2F inv 175 \$AF offset 376
KEYCODE 72 \$48 uns 8 \$08 SHIFT-O



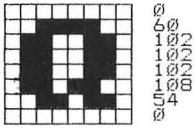
ATASCII 84 \$54 inv 212 \$D4
ML — inv —
ICODE 52 \$34 inv 180 \$B4 offset 416
KEYCODE 109 \$6D uns 45 \$2D SHIFT-T



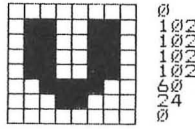
ATASCII 80 \$50 inv 208 \$D0
ML BVC inv BNE
ICODE 48 \$30 inv 176 \$B0 offset 384
KEYCODE 74 \$4A uns 10 \$0A SHIFT-P



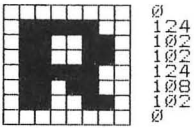
ATASCII 85 \$55 inv 213 \$D5
ML EOR z,X inv CMP z,X
ICODE 53 \$35 inv 181 \$B5 offset 424
KEYCODE 75 \$4B uns 11 \$0B SHIFT-U



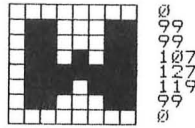
ATASCII 81 \$51 inv 209 \$D1
ML EOR (ind),Y inv CMP (ind),Y
ICODE 49 \$31 inv 177 \$B1 offset 392
KEYCODE 111 \$6F uns 47 \$2F SHIFT-Q



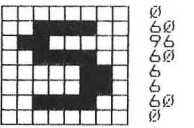
ATASCII 86 \$56 inv 214 \$D6
ML LSR z,X inv DEC abs
ICODE 54 \$36 inv 182 \$B6 offset 432
KEYCODE 80 \$50 uns 16 \$10 SHIFT-V



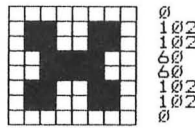
ATASCII 82 \$52 inv 210 \$D2
ML — inv —
ICODE 50 \$32 inv 178 \$B2 offset 400
KEYCODE 104 \$68 uns 40 \$28 SHIFT-R



ATASCII 87 \$57 inv 215 \$D7
ML — inv —
ICODE 55 \$37 inv 183 \$B7 offset 440
KEYCODE 110 \$6E uns 46 \$2E SHIFT-W

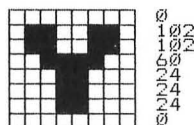


ATASCII 83 \$53 inv 211 \$D3
ML — inv —
ICODE 51 \$33 inv 179 \$B3 offset 408
KEYCODE 126 \$7E uns 62 \$3E SHIFT-S

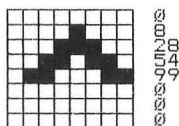


ATASCII 88 \$58 inv 216 \$D8
ML CLI inv CLD
ICODE 56 \$38 inv 184 \$B8 offset 448
KEYCODE 86 \$56 uns 22 \$16 SHIFT-X

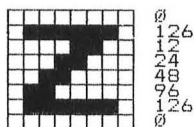
Table 1. Atari Character Set (continued)



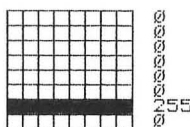
ATASCII	89	\$59	inv	217	\$D9
ML EOR	abs,Y		inv	CMP	abs,X
ICODE	57	\$39	inv	185	\$B9 offset 456
KEYCODE	107	\$66	uns	43	\$2B SHIFT-Y



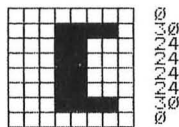
ATASCII	94	\$5E	inv	222	\$DE
ML LSR	abs,X		inv	DEC	abs,X
ICODE	62	\$3E	inv	190	\$BE offset 496
KEYCODE	71	\$47	uns	7	\$07 SHIFT-*



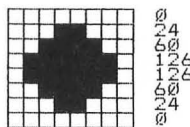
ATASCII	90	\$5A	inv	218	\$DA
ML	—		inv	—	
ICODE	58	\$3A	inv	186	\$BA offset 464
KEYCODE	87	\$57	uns	23	\$17 SHIFT-z



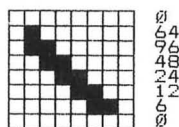
ATASCII	95	\$5F	inv	223	\$DF
ML	—		inv	—	
ICODE	63	\$3F	inv	191	\$BF offset 504
KEYCODE	78	\$4E	uns	14	\$0E SHIFT- [hyphen]



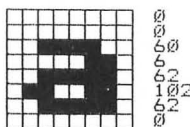
ATASCII	91	\$5B	inv	219	\$DB
ML	—		inv	—	
ICODE	59	\$3B	inv	187	\$BB offset 472
KEYCODE	96	\$60	uns	32	\$20 SHIFT-,



ATASCII	96	\$60	inv	224	\$E0
ML RTS			inv	CPX	#
ICODE	96	\$60	inv	224	\$E0 offset 768
KEYCODE	162	\$A2	uns	34	\$22 CONTROL-



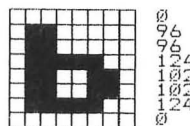
ATASCII	92	\$5C	inv	220	\$DC
ML	—		inv	—	
ICODE	60	\$3C	inv	188	\$BC offset 480
KEYCODE	70	\$46	uns	6	\$06 SHIFT-+



ATASCII	97	\$61	inv	225	\$E1
ML ADC	(ind,X)		inv	SBC	(ind,X)
ICODE	97	\$61	inv	225	\$E1 offset 776
KEYCODE	63	\$3F	uns	63	\$3F A



ATASCII	93	\$5D	inv	221	\$DD
ML EOR	abs,X		inv	CMP	abs,X
ICODE	61	\$3D	inv	189	\$BD offset 488
KEYCODE	98	\$62	uns	34	\$22 SHIFT-



ATASCII	98	\$62	inv	226	\$E2
ML	—		inv	—	
ICODE	98	\$62	inv	226	\$E2 offset 784
KEYCODE	21	\$15	uns	21	\$15 B

Table 1. Atari Character Set (continued)

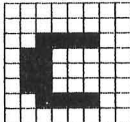
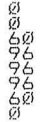
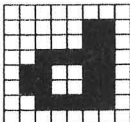
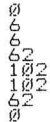
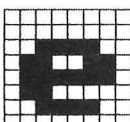
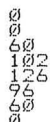

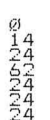
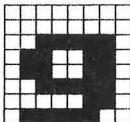
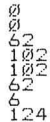
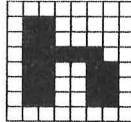
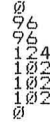
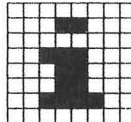
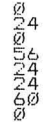
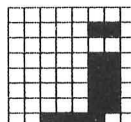
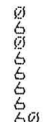
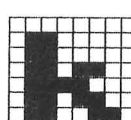
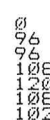
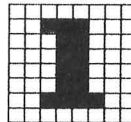
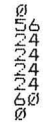
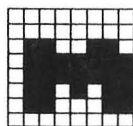
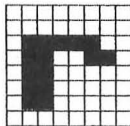
	
ATASCII 99	\$63 inv 227 \$E3
ML —	inv —
ICODE 99	\$63 inv 227 \$E3 offset 792
KEYCODE 18	\$12 uns 18 \$12 C
	
ATASCII 100	\$64 inv 228 \$E4
ML —	inv CPX z
ICODE 100	\$64 inv 228 \$E4 offset 800
KEYCODE 58	\$3A uns 58 \$3A D
	
ATASCII 101	\$65 inv 229 \$E5
ML ADC z	inv SBC z
ICODE 101	\$65 inv 229 \$E5 offset 808
KEYCODE 42	\$2A uns 42 \$2A E
	
ATASCII 102	\$66 inv 230 \$E6
ML ROR z	inv INC z
ICODE 102	\$66 inv 230 \$E6 offset 816
KEYCODE 56	\$38 uns 56 \$38 F
	
ATASCII 103	\$67 inv 231 \$E7
ML —	inv —
ICODE 103	\$67 inv 231 \$E7 offset 824
KEYCODE 61	\$3D uns 61 \$3D G
	
ATASCII 104	\$68 inv 232 \$E8
ML PLA	inv INX
ICODE 104	\$68 inv 232 \$E8 offset 832
KEYCODE 57	\$39 uns 57 \$39 H
	
ATASCII 105	\$69 inv 233 \$E9
ML ADC #	inv SBC #
ICODE 105	\$69 inv 233 \$E9 offset 840
KEYCODE 13	\$0D uns 13 \$0D I
	
ATASCII 106	\$6A inv 234 \$EA
ML ROR A	inv NOP
ICODE 106	\$6A inv 234 \$EA offset 848
KEYCODE 1	\$01 uns 1 \$01 J
	
ATASCII 107	\$6B inv 235 \$EB
ML —	inv —
ICODE 107	\$6B inv 235 \$EB offset 856
KEYCODE 5	\$05 uns 5 \$05 K
	
ATASCII 108	\$6C inv 236 \$EC
ML JMP (ind)	inv CPX abs
ICODE 108	\$6C inv 236 \$EC offset 864
KEYCODE 0	\$00 uns 0 \$00 L

Table 1. Atari Character Set (continued)



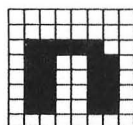
0
0
1 2
1 2
1 2
1 2
9 9
0

ATASCII 109 \$6D inv 237 \$ED
ML ADC abs inv SBC abs
ICODE 109 \$6D inv 237 \$ED offset 872
KEYCODE37 \$25 uns 37 \$25 M



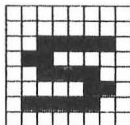
0
0
1 2 4
1 0 2
9 6
9 6
9 6
0

ATASCII 114 \$72 inv 242 \$F2
ML — inv —
ICODE 114 \$72 inv 242 \$F2 offset 912
KEYCODE40 \$28 uns 40 \$28 R



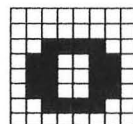
0
0
1 2 4
1 0 2
1 0 2
1 0 2
1 0 2
1 0 2
0

ATASCII 110 \$6E inv 238 \$EE
ML ROR abs inv INC abs
ICODE 110 \$6E inv 238 \$EE offset 880
KEYCODE35 \$23 uns 35 \$23 N



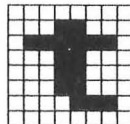
0
0
6 2
6 6
9 6
6 6
6 6
1 2 4
0

ATASCII 115 \$73 inv 243 \$F3
ML — inv —
ICODE 115 \$73 inv 242 \$F3 offset 920
KEYCODE62 \$3E uns 62 \$3E S



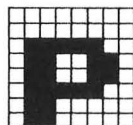
0
6 0
1 0 2
1 0 2
1 0 2
1 0 2
6 0
0

ATASCII 111 \$6F inv 239 \$EF
ML — inv —
ICODE 111 \$6F inv 239 \$EF offset 888
KEYCODE8 \$08 uns 8 \$08 O



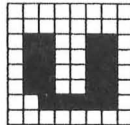
0
2 4
1 2 6
1 2 4
2 4
2 4
1 4
0

ATASCII 116 \$74 inv 244 \$F4
ML — inv —
ICODE 116 \$74 inv 244 \$F4 offset 928
KEYCODE45 \$2D uns 45 \$2D T



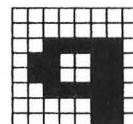
0
0
1 2 4
1 0 2
1 0 2
1 2 4
9 6
9 6

ATASCII 112 \$70 inv 240 \$F0
ML BVS inv BEQ
ICODE 112 \$70 inv 240 \$F0 offset 896
KEYCODE10 \$0A uns 10 \$0A P



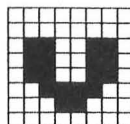
0
0
1 0 2
1 0 2
1 0 2
1 0 2
6 2
0

ATASCII 117 \$75 inv 245 \$F5
ML ADC z,X inv SBC z,X
ICODE 117 \$75 inv 245 \$F5 offset 936
KEYCODE11 \$0B uns 11 \$0B U



0
0
6 2
1 0 2
1 0 2
6 2
6 6
0

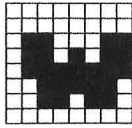
ATASCII 113 \$71 inv 241 \$F1
ML ADC (ind),Y inv SBC (ind),Y
ICODE 113 \$71 inv 241 \$F1 offset 904
KEYCODE47 \$2F uns 47 \$2F Q



0
0
1 0 2
1 0 2
1 0 2
6 0
2 4
0

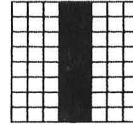
ATASCII 118 \$76 inv 246 \$F6
ML ROR z,X inv INC z,X
ICODE 118 \$76 inv 246 \$F6 offset 944
KEYCODE16 \$10 uns 16 \$10 V

Table 1. Atari Character Set (continued)



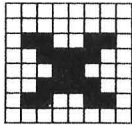
0
0
9 9
1 0 7
1 2 7
6 2
5 4
0

ATASCII 119 \$77 inv 247 \$F7
ML — inv —
ICODE 119 \$77 inv 247 \$F7 offset 952
KEYCODE 46 \$2E uns 46 \$2E W



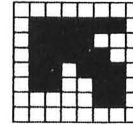
2 4
2 4
2 4
2 4
2 4
2 4
2 4
2 4

ATASCII 124 \$7C inv 252 \$FC
ML — inv —
ICODE 124 \$7C inv 252 \$FC offset 992
KEYCODE 79 \$4F uns 15 \$0F SHIFT- =



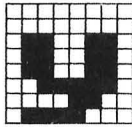
0
0
1 0 2
6 0
2 4
6 0
1 0 2
0

ATASCII 120 \$78 inv 248 \$F8
ML SEI inv SED
ICODE 120 \$78 inv 248 \$F8 offset 960
KEYCODE 22 \$16 uns 22 \$16 X



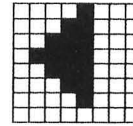
0
1 2 6
1 2 0
1 2 4
1 1 0
1 0 2
6
0

ATASCII 125 \$7D inv 253 \$FD CLEAR
ML ADC abs,X inv SBC abs,X BUZZER
[CONTROL- 2]
ICODE 125 \$7D inv 253 \$FD offset 1000
KEYCODE 118 \$76 uns 54 \$36 {ESC} SHIFT- <



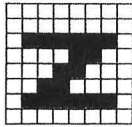
0
0
1 0 2
1 0 2
1 0 2
6 2
1 2 0
1 2 0

ATASCII 121 \$79 inv 249 \$F9
ML ADC abs,Y inv SBC abs,Y
ICODE 121 \$79 inv 249 \$F9 offset 968
KEYCODE 43 \$2B uns 43 \$2B Y



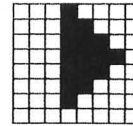
0
4
6
0 6
0 6
4
0
0

ATASCII 126 \$7E inv 254 \$FE DELETE
ML ROR abs,X inv INC abs,X DELETE
RIGHT
ICODE 126 \$7E inv 254 \$FE offset 1008
KEYCODE 52 \$34 uns 52 \$34 {ESC} DELETE



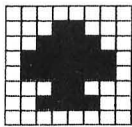
0
0
1 2 6
1 2
2 4
4 8
1 2 6
0

ATASCII 122 \$7A inv 250 \$FA
ML — inv —
ICODE 122 \$7A inv 250 \$FA offset 976
KEYCODE 23 \$17 uns 23 \$17 z



1 6
2 4
0 8
0 8
0 8
4 4
1 6
0

ATASCII 127 \$7F inv 255 \$FF TAB
ML — inv — INSERT
CHARACTER
ICODE 127 \$7F inv 255 \$FF offset 1016
KEYCODE 44 \$2C uns 44 \$2C {ESC} TAB



0
2 4
6 0
1 2 6
1 2 6
2 4
6 0
0

ATASCII 123 \$7B inv 251 \$FB
ML — inv —
ICODE 123 \$7B inv 251 \$FB offset 984
KEYCODE 130 \$82 uns 2 \$02 CONTROL-;

Table 2. Internal Code Index, ICODE: ATASCII

0:	32	41:	73	82:	18	123:	123	164:	196
1:	33	42:	74	83:	19	124:	124	165:	197
2:	34	43:	75	84:	20	125:	125	166:	198
3:	35	44:	76	85:	21	126:	126	167:	199
4:	36	45:	77	86:	22	127:	127	168:	200
5:	37	46:	78	87:	23	128:	160	169:	201
6:	38	47:	79	88:	24	129:	161	170:	202
7:	39	48:	80	89:	25	130:	162	171:	203
8:	40	49:	81	90:	26	131:	163	172:	204
9:	41	50:	82	91:	27	132:	164	173:	205
10:	42	51:	83	92:	28	133:	165	174:	206
11:	43	52:	84	93:	29	134:	166	175:	207
12:	44	53:	85	94:	30	135:	167	176:	208
13:	45	54:	86	95:	31	136:	168	177:	209
14:	46	55:	87	96:	96	137:	169	178:	210
15:	47	56:	88	97:	97	138:	170	179:	211
16:	48	57:	89	98:	98	139:	171	180:	212
17:	49	58:	90	99:	99	140:	172	181:	213
18:	50	59:	91	100:	100	141:	173	182:	214
19:	51	60:	92	101:	101	142:	174	183:	215
20:	52	61:	93	102:	102	143:	175	184:	216
21:	53	62:	94	103:	103	144:	176	185:	217
22:	54	63:	95	104:	104	145:	177	186:	218
23:	55	64:	0	105:	105	146:	178	187:	219
24:	56	65:	1	106:	106	147:	179	188:	220
25:	57	66:	2	107:	107	148:	180	189:	221
26:	58	67:	3	108:	108	149:	181	190:	222
27:	59	68:	4	109:	109	150:	182	191:	223
28:	60	69:	5	110:	110	151:	183	192:	128
29:	61	70:	6	111:	111	152:	184	193:	129
30:	62	71:	7	112:	112	153:	185	194:	130
31:	63	72:	8	113:	113	154:	186	195:	131
32:	64	73:	9	114:	114	155:	187	196:	132
33:	65	74:	10	115:	115	156:	188	197:	133
34:	66	75:	11	116:	116	157:	189	198:	134
35:	67	76:	12	117:	117	158:	190	199:	135
36:	68	77:	13	118:	118	159:	191	200:	136
37:	69	78:	14	119:	119	160:	192	201:	137
38:	70	79:	15	120:	120	161:	193	202:	138
39:	71	80:	16	121:	121	162:	194	203:	139
40:	72	81:	17	122:	122	163:	195	204:	140

205: 141	216: 152	227: 227	238: 238	249: 249
206: 142	217: 153	228: 228	239: 239	250: 250
207: 143	218: 154	229: 229	240: 240	251: 251
208: 144	219: 155	230: 230	241: 241	252: 252
209: 145	220: 156	231: 231	242: 242	253: 253
210: 146	221: 157	232: 232	243: 243	254: 254
211: 147	222: 158	233: 233	244: 244	255: 255
212: 148	223: 159	234: 234	245: 245	
213: 149	224: 224	235: 235	246: 246	
214: 150	225: 225	236: 236	247: 247	
215: 151	226: 226	237: 237	248: 248	

Table 3. Machine Language Index

MNEMONIC	ATASCI	MNEMONIC	ATASCI	MNEMONIC	ATASCI
ADC #	105	BIT abs	44	CPY abs	204
ADC abs	109	BIT z	36	CPY z	196
ADC abs, X	125	BMI	48	DEC abs	214
ADC abs, Y	121	BNE	208	DEC abs, X	222
ADC (ind, X)	97	BPL	16	DEC z	198
ADC (ind), Y	113	BRK	0	DEC z, X	206
ADC z	101	BVC	80	DEX	202
ADC z, X	117	BVS	112	DEY	136
AND #	41	CLC	24	EOR #	73
AND abs	45	CLD	216	EOR abs	77
AND abs, X	61	CLI	88	EOR abs, X	93
AND abs, Y	57	CLV	184	EOR abs, Y	89
AND (ind, X)	33	CMP #	201	EOR (ind, X)	65
AND (ind), Y	49	CMP abs	205	EOR (ind), Y	81
AND z	37	CMP abs, X	221	EOR z	69
AND z, X	53	CMP abs, Y	217	EOR z, X	85
ASL A	10	CMP (ind, X)	193	INC abs	238
ASL abs	14	CMP (ind), Y	209	INC abs, X	254
ASL abs, X	30	CMP z	197	INC z	230
ASL z	6	CMP z, X	213	INC z, X	246
ASL z, X	22	CPX #	224	INX	232
BCC	144	CPX abs	236	INY	200
BCS	176	CPX z	228	JMP abs	76
BEQ	240	CPY #	192	JMP (ind)	108

JSR abs	32	ORA abs	13	SBC abs, Y	249
LDA #	169	ORA abs, X	29	SBC (ind, X)	225
LDA abs	173	ORA abs, Y	25	SBC (ind), Y	241
LDA abs, X	189	ORA (ind, X)	1	SBC z	229
LDA abs, Y	185	ORA (ind), Y	17	SBC z, X	245
LDA (ind, X)	161	ORA z	5	SEC	56
LDA (ind), Y	177	ORA z, X	21	SED	248
LDA z	165	PHA	72	SEI	120
LDA z, X	181	PHP	8	STA abs	141
LDX #	162	PLA	104	STA abs, X	157
LDX abs	174	PLP	40	STA abs, Y	153
LDX abs, Y	190	ROL A	42	STA (ind, X)	129
LDX z	166	ROL abs	46	STA (ind), Y	145
LDX z, Y	182	ROL abs, X	62	STA z	133
LDY #	160	ROL z	38	STA z, X	149
LDY abs	172	ROL z, X	54	STX abs	142
LDY abs, X	188	ROR A	106	STX z	134
LDY z	164	ROR abs	110	STX z, Y	150
LDY z, X	180	ROR abs, X	126	STY abs	140
LSR A	74	ROR z	102	STY z	132
LSR abs	78	ROR z, X	118	STY z, X	148
LSR abs, X	94	RTI	64	TAX	170
LSR z	70	RTS	96	TAY	168
LSR z, X	86	SBC #	233	TSX	186
NOP	234	SBC abs	237	TXA	138
ORA #	9	SBC abs, X	253	TXS	154
				TYA	152

Table 4. Keyboard Code Index, KEYCODE: ATASCII

Key	Single key press	SHIFT and key	CONTROL and key	SHIFT and CONTROL and key
L	0: 108	64: 76	128: 12	L
J	1: 106	65: 74	129: 10	J
;	2: 59	66: 58	130: 123	;
K	5: 107	69: 75	133: 11	K
+	6: 43	70: 92	134: 30	+
*	7: 42	71: 94	135: 31	*
O	8: 111	72: 79	136: 15	O
P	10: 112	74: 80	138: 16	P
U	11: 117	75: 85	139: 21	U
RET	12: c	76: c	140: c	RET
I	13: 105	77: 73	141: 9	I
-	14: 45	78: 95	142: 28	-
=	15: 61	79: 124	143: 29	=
V	16: 118	80: 86	144: 90	V
C	18: 99	82: 67	146: 3	C
B	21: 98	85: 66	149: 2	B
X	22: 120	86: 88	150: 24	X
Z	23: 122	87: 90	151: 26	Z

Key	Single key press	SHIFT and key	CONTROL and key	SHIFT and CONTROL and key
4	24: 52	88: 36	152: <i>a</i>	216: <i>a</i> 4
3	26: 51	90: 35	154: <i>a</i>	218: <i>a</i> 3
6	27: 54	91: 38	155: <i>a</i>	219: <i>a</i> 6
ESC	28: 27	92: 27	156: 27	220: <i>a</i> ESC
5	29: 53	93: 37	157: <i>a</i>	221: <i>a</i> 5
2	30: 50	94: 34	158: <i>c</i> 253	222: <i>a</i> 2
1	31: 49	95: 33	159: <i>b</i>	223: <i>a</i> 1
' SPC	32: 44	96: 91	160: 0	224: <i>a</i> ' SPC
N	33: 32	97: 32	161: 32	225: <i>a</i> N
M	35: 110	99: 78	163: 14	227: <i>a</i> M
/	37: 109	101: 77	165: 13	229: <i>a</i> /
INV	38: 47	102: 63	166: <i>a</i>	230: <i>a</i> INV
	39: <i>d</i>	103: <i>d</i>	167: <i>d</i>	231: <i>a</i>
R	40: 114	104: 82	168: 18	232: <i>a</i> R
E	42: 101	106: 69	170: 5	234: <i>a</i> E
Y	43: 121	107: 89	171: 25	235: <i>a</i> Y
TAB	44: 127	108: <i>c</i> 159	172: <i>c</i> 158	236: <i>a</i> TAB
T	45: 116	109: 84	173: 20	237: <i>a</i> T
W	46: 119	110: 87	174: 23	238: <i>a</i> W
Q	47: 113	111: 81	175: 17	239: <i>a</i> Q

Key	Single key press	SHIFT and key	CONTROL and key	SHIFT and CONTROL and key
9	48: 57	112: 40	176: a	240: a 9
0	50: 48	114: 41	178: a	242: a 0
7	51: 55	115: 39	179: a	243: a 7
DEL	52: 126	116: c	180: c	244: a DEL
8	53: 56	117: 64	181: a	245: a 8
<	54: 60	118: 125	182: 125	246: a <
>	55: 62	119: c	183: c	247: a >
F	56: 102	120: 70	184: 6	248: a F
H	57: 104	121: 72	185: 8	249: a H
D	58: 100	122: 68	186: 4	250: a D
CAP	60: e	124: e	188: e	252: a CAP
G	61: 103	125: 71	189: 7	253: a G
S	62: 115	126: 83	190: 19	254: a S
A	63: 97	127: 65	191: 1	255: a A

a No ASCII value assigned.

b CONTROL-I interrupt; cannot be read at 764

c Editing function; inverse ATASCII only (subtract 128 to find ASCII entry)

d Inverse video key (Atari logo key)

e CAPS-LOWER key

Note: All missing numbers are invalid keyboard codes, which will never be found in location 764.

B

Appendix

A Beginner's Guide to Typing In Programs

What Is a Program?

A computer cannot perform any task by itself. Like a car without gas, a computer has *potential*, but without a program, it isn't going anywhere. Most of the programs published in this book are written in a computer language called BASIC. Atari 8K BASIC is easy to learn.

BASIC Programs

Computers can be picky. Unlike the English language, which is full of ambiguities, BASIC usually has only one "right way" of stating something. Every letter, character, or number is significant. A common mistake is substituting a letter such as "O" for the numeral "0", a lowercase "l" for the numeral "1", or an uppercase "B" for the numeral "8". Also, you must enter all punctuation such as colons and commas just as they appear in the book. Spacing can be important. To be safe, type in the listings *exactly* as they appear.

Braces and Special Characters

The exception to this typing rule is when you see the braces, such as "{DOWN}". Anything within a set of braces is a special character or characters that cannot easily be listed on a printer. When you come across such a special statement, refer to Appendix C, "How to Type in Programs."

About DATA Statements

Some programs contain a section or sections of DATA statements. These lines provide information needed by the program. Some DATA statements contain actual programs (called machine language); others contain graphics codes. These lines are especially sensitive to errors.

If a single number in any one DATA statement is mistyped, your machine could “lock up,” or “crash.” The keyboard, break key, and RESET keys may all seem “dead,” and the screen may go blank. Don’t panic — no damage is done. To regain control, you have to turn off your computer, then turn it back on. This will erase whatever program was in memory, so always SAVE a copy of your program before you RUN it. If your computer crashes, you can LOAD the program and look for your mistake.

Sometimes a mistyped DATA statement will cause an error message when the program is RUN. The error message may refer to the program line that READs the data. *This error is still in the DATA statements, though.*

Get to Know Your Machine

You should familiarize yourself with your computer before attempting to type in a program. Learn the statements you use to store and retrieve programs from tape or disk. You’ll want to save a copy of your program, so that you won’t have to type it in every time you want to use it. Learn to use the machine’s editing functions. How do you change a line if you made a mistake? You can always retype the line, but you at least need to know how to backspace. Do you know how to enter inverse video, lowercase, and control characters? It’s all explained in your computer’s manuals.

A Quick Review

1. Type in the program a line at a time, in order. Press RETURN at the end of each line. Use backspace or the back arrow to correct mistakes.
2. Check the line you’ve typed against the line in the listing. You can check the entire program again if you get an error when you RUN the program.
3. Make sure you’ve entered statements in braces as the appropriate control key (see Appendix C).
4. Be sure to SAVE the program on tape or disk *before* RUNning the program.

C Appendix

How to Type In Programs

In order to make special characters, inverse video, and cursor characters easy to type in, *COMPUTE!* Magazine's Atari listing conventions are used in all the program listings in this book.

Please refer to the following tables and explanations if you come across an unusual symbol in a program listing.

Atari Conventions

Characters in inverse video will appear like: **INVERSE VIDEO**
Enter these characters with the Atari logo key, {^}.

When you see	Type	See
{CLEAR}	ESC SHIFT <	↵ Clear Screen
{UP}	ESC CTRL -	↑ Cursor Up
{DOWN}	ESC CTRL =	↓ Cursor Down
{LEFT}	ESC CTRL +	← Cursor Left
{RIGHT}	ESC CTRL *	→ Cursor Right
{BACK S}	ESC DELETE	␣ Backspace
{DELETE}	ESC CTRL DELETE	␣ Delete Character
{INSERT}	ESC CTRL INSERT	␣ Insert Character
{DEL LINE}	ESC SHIFT DELETE	␣ Delete Line
{INS LINE}	ESC SHIFT INSERT	␣ Insert Line
{TAB}	ESC TAB	␣ TAB key
{CLR TAB}	ESC CTRL TAB	␣ Clear TAB
{SET TAB}	ESC SHIFT TAB	␣ Set TAB stop
{BELL}	ESC CTRL 2	␣ Ring Buzzer
{ESC}	ESC ESC	␣ ESCape key

Graphics characters, such as CTRL-T, the ball character ● will appear as the "normal" letter enclosed in braces, e.g., {T}.

A series of identical control characters, such as 10 spaces, three cursor-lefts, or 20 CTRL-R's, will appear as {10 SPACES}, {3 LEFT}, {20 R}, etc. If the character in braces is in inverse video, that character or characters should be entered with the Atari logo key. For example, {■} means to enter a reverse-field heart with CTRL-comma, {5■} means to enter five inverse-video CTRL-U's.

Index

- ADR function 217
- alternate character sets, in "Super TextPlot" 142
- angular orientations, in "Super TextPlot" 142
- ANTIC (A) command, in "SuperFont Plus" 129
- ANTIC chip 6
- ANTIC 4/5 character set, in "SuperFont Plus" 127, 128
- ANTIC 4 mode 127, 245
- ANTIC 5 mode 127, 245
- Assembler Editor 248, 249
- Assembler Editor manual 248
- Atari 400 233
- Atari 800 233
- Atari 1200XL 233-47
 - incompatibilities 233, 242
 - key definition 235-36
 - memory map 233-47
 - new graphics modes 245-46
 - OS 244
- Atari 1200XL Operating System Manual* 240, 247
- Atari BASIC graphics capabilities 201
- Atari BASIC Reference Manual* 38, 225
- Atari character set 275-98
 - tables 279-98
- Atari Personal Computer System Hardware Manual* 37-38, 48, 201
- "Atari Verify" utility 165-66
- Atari XL models *see* XL models
- ATASCII 5-6, 142, 225, 248-49, 275, 292
- AUDCTL register 38-39, 45
- audio control registers 45
- audio frequency registers 45
- "Automate" 167-73
- automated system commands 167-73
- AUTORUN.SYS file *see* "Automate"
- back-arrow, as paragraph delimiter in "Scriptor" 105-6
- BASIC, extensions to 174
- BASIC cartridge 249
- "Beginner's Keyboard" 55-56
- blinking characters 27-30
- braces, in program listings 299
- BREAK key 175
- bubble sort 259
- "CalCalc" 87-93
- CAPS/LOWR key 8, 13, 14, 15, 17
- cartridge, advantage for OS 175
- "Castle Quest" 94-101
- CDRMA2 register 28
- CDTMA2 register 28
- CDTMV2 register 28
- CHACT register and inverse video 27-28
- CHBAS vector 142
- CIO (Central Input/Output) 176, 179
- circles 153-60
 - difficulties in drawing 153
 - potential method 156-57
 - sines and cosines method 154-55
 - square root method 155-56
 - techniques 154
- code conversions 10-11
 - ATASCII-ICODE conversions 10-11
 - KEYCODE-ATASCII or ICODE 11
- Color Change mode, in "SuperFont Plus" 130
- color rotation in P/M graphics 202
- COMPUTE!'s First Book of Atari* 67
- COMPUTE!'s First Book of Atari Graphics* 127, 142, 202
- COMPUTE!'s Mapping the Atari* 233, 246
- COMPUTE!'s Second Book of Atari* 165
- COMPUTE!'s Second Book of Atari Graphics* 217
- CONTROL-DELETE key 12
- CONTROL-INSERT key 12
- CONTROL-lock key 8
- CONTROL-1 key 10
- CONTROL-2 key 12
- CONTROL-TAB key 12
- "The Cruncher" 225-27
- CTRL key, in "Scriptor" 105, 106, 107
- cursor 103
 - control with joystick 163-64
- cursor codes 163
- DATA statement 299
- debounce 17
- De Re Atari* 202
- DIMension statement 31-32
- DIR command 179
- disk files 104
 - naming rules 107-8
- display list, relocating 201
- Dvorak keyboard layout 4, 8, 16
- 8-bit note table 49-50
- "Elementary Numbers" 66-73
- exponential operator 3
- exponents 3
 - correcting inaccuracy of 3
- F1 key 9, 13
 - on 1200XL 234

- F2 key 9, 13
- F3 key 9, 13
- F4 key 9, 13
- Fn keys 9
- FOR/NEXT loops, inaccurate for timing 22
- GET 23
- GRAPHICS 0 mode 58, 129, 228
- GRAPHICS 1 mode 129
- GRAPHICS 2 mode 74, 129
- GRAPHICS 7 mode 201
- GRAPHICS 12 mode 245-46
- GRAPHICS 13 mode 245-46
- GRAPHICS 14 mode 245-46
- GRAPHICS 15 mode 245-46
- HELP key 9, 13
- ICODE 5, 6, 10-11, 227, 292-93
- internal code *see* ICODE
- Internal Code Index 292-93
- interrupts 9
- INT function 3
- jiffy 228
- joystick 67
 - for cursor control 163-64
- joystick codes 163
- keyboard code *see also* KEYCODE
- KEYCODE 4, 5, 7, 8, 10-11, 277, 296-98
 - on XL models 9
- keys, customization 4-5
- "Laser Gunner II" 216-24
- LOCK command 178, 179
- machine language, merging with
 - BASIC
 - discussion 248-57
 - safe memory 250-51
- Machine Language Index 294-95
- matrix wastes memory 225
- Mini-DOS, in "Scriptor" 107
- music, 16-bit 45-51
- Operating System, *see* OS
- OS, defined 174-75
- PEEK and POKE, sometimes faster than
 - conventional commands 228
- peripherals 175-76
- Player/Missile graphics 201
 - explosions 205
 - fast motion 216
 - player definition 202
- POKE to RAMTOP 202
- TRAPs 204-5
- PLOT command 225
- PRINT mode, in "SuperFont Plus" 129
- purging diskettes 195-98
- Qwerty keyboard layout 4, 8
- RAMTOP
 - and strings 217
 - changing 201
- RENAME command 178, 179
- "Renumber Plus" utility 191-94
- roots, exponential 3
- RUBOUT key 102
- SAVE, importance of 300
- SAVE, important in "Scriptor" program 105
- SCRATCH command 178, 179
- screen editing characters 278
- screen RAM, relocating 201
- "Scriptor" 102-23
 - customizing 110
 - edit commands 111
 - formatting commands 112
 - Mini-DOS in 107
 - RETURN and 105
 - SAVE, important with 105
 - sheet feeding 109-10
 - text formatting 108-9
- selection and exchange sort 259
- selection sort 259
- self-modifying code, in "Standings" program 76
- SHIFT-CONTROL combinations 9
- SHIFT-DELETE key 12
- SHIFT-INSERT key 12
- SHIFT-lock key 8
- SHIFT-RETURN key 12
- SHIFT-TAB key 12
- 16-bit dividers 47
- 16-bit note table 49-50
- 16-bit sound 48, 50-51
- 6502 machine language 167, 276
- sort, defined 258
- sort routine, in "Standings" program 75
- sort utility, machine language 258-71
 - faster than BASIC 260
 - instructions 261-62
 - options 262
- sound 37-44
 - difficulties with 37
- "Sound Experimenter" 39-44
- SOUND instruction 37, 50, 225
- "Spelling Quiz" 57-65
- SQR function 3
- "Standings" 74-86

- "Starshot" 201-15
- string arrays 31-33
 - fast initialization 31
- strings
 - and renumbering 191
 - holding Player/Missile data 217
 - machine language in 252-53
 - take less space than matrices 225
- "SuperFont" 127
 - commands 128-29
- "SuperFont Plus" 127-41
 - new commands 129
- "Super TextPlot" 142-52
 - alternate character sets in 142
 - angular orientations in 142, 147
 - applications 148
 - cautions with 146-47
 - loading 147-48
 - mathematics of 143-44
 - parameters 144-46
- text formatting, in "Scriptor" 108-9
- "TextPlot" 142
- timer 22-26
- times, sample 24
- time test programs 229-30
- timing, accurate 23-26
- tokens, BASIC 191
- TRAP 32-33, 196, 204-5, 227
- tuning inaccuracy 46
 - reduced by 16-bit dividers 47
- UNLOCK command 178, 179
- USR function 144
- Vertical Blank Interrupt 217-19
- vertical blank period 27, 29, 163
- wedge 176-77
- "The Wedge" 174-90
- word processing concepts 102-3
- XIO 196
- XL models 7, 9, 13
- young children, computers and 66

COMPUTE! Books

P.O. Box 5406 Greensboro, NC 27403

Ask your retailer for these **COMPUTE! Books**. If he or she has sold out, order directly from **COMPUTE!**.

For Fastest Service
Call Our **TOLL FREE US Order Line**
800-334-0868
In NC call **919-275-9809**

Quantity	Title	Price	Total
_____	COMPUTE!'s First Book of Atari (00-0)	\$12.95	_____
_____	COMPUTE!'s Second Book of Atari (06-X)	\$12.95	_____
_____	COMPUTE!'s Third Book of Atari (18-3)	\$12.95	_____
_____	COMPUTE!'s First Book of Atari Graphics (08-6)	\$12.95	_____
_____	COMPUTE!'s Second Book of Atari Graphics (28-0)	\$12.95	_____
_____	Mapping the Atari (09-4)	\$14.95	_____
_____	COMPUTE!'s First Book of Atari Games (14-0)	\$12.95	_____
_____	The Atari BASIC Source Book (15-9)	\$12.95	_____
_____	Inside Atari DOS (02-7)	\$19.95	_____
_____	COMPUTE!'s Atari Collection, Volume 1 (79-5)	\$12.95	_____
_____	Machine Language for Beginners (11-6)	\$14.95	_____
_____	Second Book of Machine Language (53-1)	\$14.95	_____
_____	Computing Together: A Parent and Teacher's Guide to Using Computers with Young Children (51-5)	\$12.95	_____
_____	Personal Telecomputing (47-7)	\$12.95	_____
_____	Home Energy Applications on Your Personal Computer (10-8)	\$14.95	_____

Add \$2.00 shipping and handling. Outside US add \$5.00 air mail or \$2.00 surface mail.

Please add shipping & handling for each book ordered. _____

Total enclosed or to be charged _____

All orders must be prepaid (money order, check, or charge). All payments must be in US funds. NC residents add 4½% sales tax.

☐ Payment enclosed Please charge my: ☐ Visa ☐ MasterCard
☐ American Express

Acct. No. _____ Exp. Date _____

Name _____

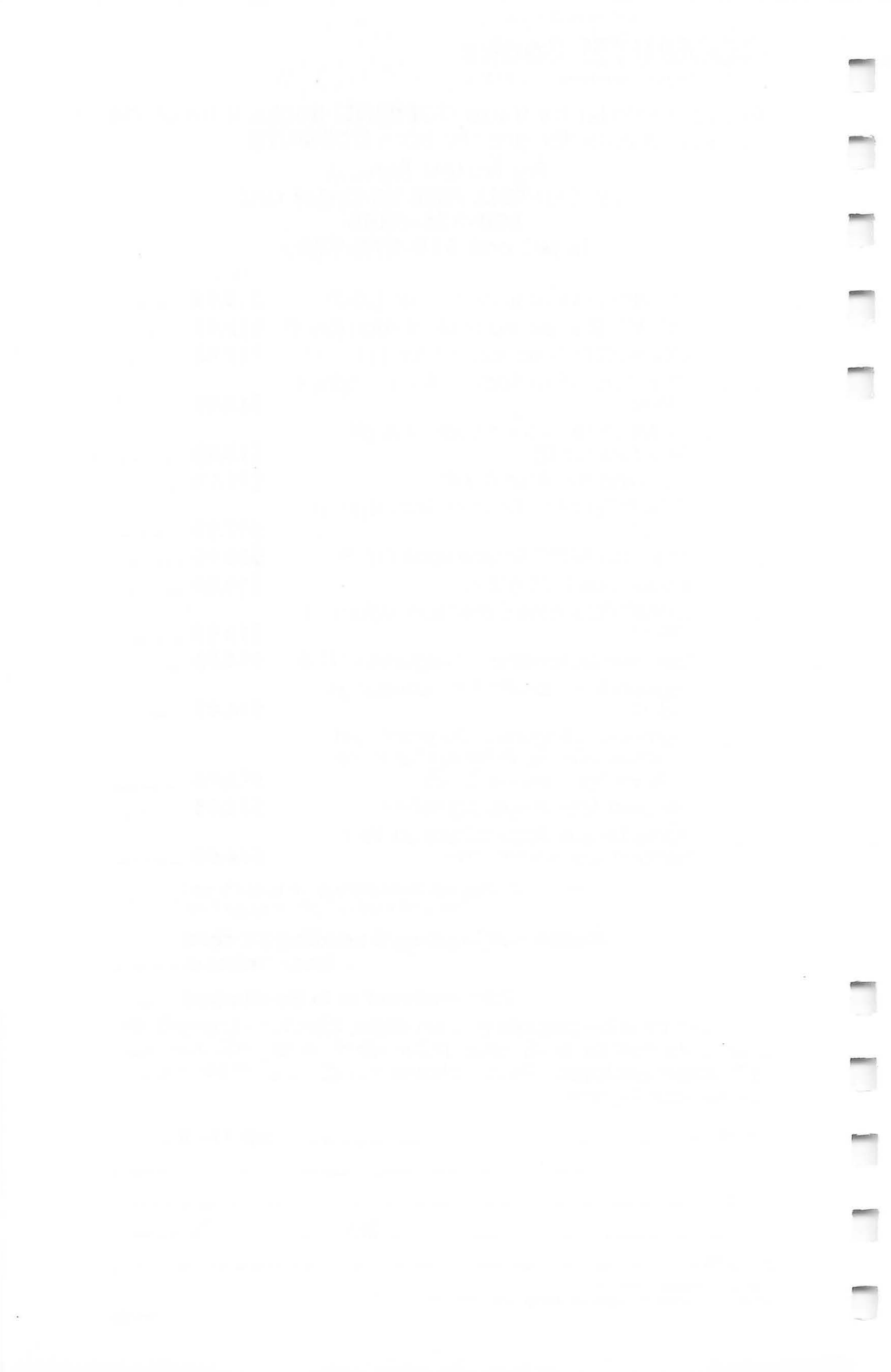
Address _____

City _____ State _____ Zip _____

Country _____

*Allow 4-5 weeks for delivery.

Prices and availability subject to change without notice.



COMPUTE! Books

Ask your retailer for these **COMPUTE! Books** or order directly from **COMPUTE!**.

Call toll free (in US) **800-334-0868** (in NC 919-275-9809) or write COMPUTE! Books, P.O. Box 5406, Greensboro, NC 27403.

Quantity	Title	Price*	Total
_____	Machine Language for Beginners (11-6)	\$14.95	_____
_____	The Second Book of Machine Language (53-1)	\$14.95	_____
_____	COMPUTE!'s Guide to Adventure Games (67-1)	\$12.95	_____
_____	Computing Together: A Parents & Teachers Guide to Computing with Young Children (51-5)	\$12.95	_____
_____	Personal Telecomputing (47-7)	\$12.95	_____
_____	BASIC Programs for Small Computers (38-8)	\$12.95	_____
_____	Programmer's Reference Guide to the Color Computer (19-1)	\$12.95	_____
_____	Home Energy Applications (10-8)	\$14.95	_____
_____	The Home Computer Wars: An Insider's Account of Commodore and Jack Tramiel		_____
	Hardback (75-2)	\$16.95	_____
	Paperback (78-7)	\$ 9.95	_____
_____	The Book of BASIC (61-2)	\$12.95	_____
_____	Every Kid's First Book of Robots and Computers (05-1)	\$ 4.95†	_____
_____	The Beginner's Guide to Buying a Personal Computer (22-1)	\$ 3.95†	_____

* Add \$2.00 per book for shipping and handling.

† Add \$1.00 per book for shipping and handling.

Outside US add \$5.00 air mail or \$2.00 surface mail.

Shipping & handling: \$2.00/book _____
Total payment _____

All orders must be prepaid (check, charge, or money order).

All payments must be in US funds.

NC residents add 4.5% sales tax.

☐ Payment enclosed.

Charge ☐ Visa ☐ MasterCard ☐ American Express

Acct. No. _____ Exp. Date _____

Name _____

Address _____

City _____ State _____ Zip _____

*Allow 4-5 weeks for delivery.

Prices and availability subject to change.

Current catalog available upon request.

If you've enjoyed the articles in this book, you'll find the same style and quality in every monthly issue of **COMPUTE!** Magazine. Use this form to order your subscription to **COMPUTE!**.

For Fastest Service,
Call Our **Toll-Free** US Order Line
800-334-0868
In NC call 919-275-9809

COMPUTE!

P.O. Box 5406
Greensboro, NC 27403

My Computer Is:

- ☐ Commodore 64 ☐ TI-99/4A ☐ Timex/Sinclair ☐ VIC-20 ☐ PET
☐ Radio Shack Color Computer ☐ Apple ☐ Atari ☐ Other _____
☐ Don't yet have one...

- ☐ \$24 One Year US Subscription
☐ \$45 Two Year US Subscription
☐ \$65 Three Year US Subscription

Subscription rates outside the US:

- ☐ \$30 Canada
☐ \$42 Europe, Australia, New Zealand/Air Delivery
☐ \$52 Middle East, North Africa, Central America/Air Mail
☐ \$72 Elsewhere/Air Mail
☐ \$30 International Surface Mail (lengthy, unreliable delivery)

Name _____

Address _____

City _____

State _____

Zip _____

Country _____

Payment must be in US Funds drawn on a US Bank; International Money Order, or charge card.

☐ Payment Enclosed

☐ VISA

☐ MasterCard

☐ American Express

Acc t. No. _____

Expires _____

/

COMPUTE!'s Third Book of Atari

Here is *COMPUTE!'s Third Book of Atari*, including some of the best articles, programs, and games from *COMPUTE!* Magazine, plus many more articles and programs that have never before appeared in print.

Here are just a few of the items of value in this book:

- A complete word processor program
- A spelling quiz program to help your children learn their weekly spelling words
- A complete guide to the Atari character set
- A memory map for the 1200XL
- A machine language program that allows cursor movement with a joystick
- A BASIC renumbering utility
- A very fast sorting utility written in machine language
- Articles on sound and graphics
- Three exciting games
- Plus many other useful articles

No matter whether you are an advanced programmer or just starting out, *COMPUTE!'s Third Book of Atari* has much of value to you.